(2)

USAISEC

US Army Information Systems Engineering Command
Fort Huachuca, AZ 85613-5300

U.S. ARMY INSTITUTE FOR RESEARCH
IN MANAGEMENT INFORMATION,
COMMUNICATIONS, AND COMPUTER SCIENCES

SAMeDL: Technical Report and Appendices A, B, and G

ASQB-GI-92-015

September 1992

AIRMICS
115 O'Keefe Building
Georgia Institute of Technology
Atlanta, GA 30332-0800

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188
Exp. Date: Jun 30, 1986

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS NONE
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT N/A
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S) N/A
6a. NAME OF PERFORMING ORGANIZATION / 6b. OFFICE SYMBOL *(If applicable)*	7a. NAME OF MONITORING ORGANIZATION N/A
6c. ADDRESS (City, State, and Zip Code)	7b. ADDRESS *(City, State, and ZIP Code)* N/A
8b. NAME OF FUNDING/SPONSORING ORGANIZATION Software Technology Branch, ARL / 8b. OFFICE SYMBOL *(If applicable)* AMSRL-CI-CD	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS *(City, State, and ZIP Code)* 115 O'Keefe Bldg. Georgia Institute of Technology Atlanta, GA 30332-0800	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE *(Include Security Classification)*
SAMeDL: Technical Report & Appendices A, B & G

12. PERSONAL AUTHOR(S)
MS. Deb Waterman

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT *(Year, Month, Day)*	15. PAGE COUNT
Technical Paper	FROM Apr 91 TO Sept 92	Sept 15, 1992	81

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
FIELD	GROUP	SUBGROUP	Ada Database Access, SAMeDL, Ada extension module, SQL

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

This report details the research efforts into the SQL Ada Module Database Description Language (SAMeDL). Four compilers are presented (Oracle, Informix, XDB, and Sybase) that allow Ada application programs to access database using a standard SQL query language. Copies of the compiler can be obtained from the DoD Ada Joint Program Office 703/614-0209.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL LTC David S. Stevens	22b. TELEPHONE *(Include Area Code)* (404) 894-3110	22c. OFFICE SYMBOL AMSRL-CI-CD

DD FORM 1473, 84 MAR   83 APR edition may be used until exhausted. All other editions are obsolete.   SECURITY CLASSIFICATION OF THIS PAGE

This research was performed by Statistica Inc., contract number DAKF11-91-C-0035, for the Army Institute for Research in Management Information, Communications, and Computer Sciences (AIRMICS), the RDTE organization of the U. S. Army Information Systems Engineering Command (USAISEC). This final report discusses a set of SAMeDL compilers and work enviornment that were developed during the contract. Request for copies of the compiler can be obtained from the DoD Ada Joint Program Office, 703/614/0209. This research report is not to construed as an official Army or DoD Position, unless so designated by other authorized documents. Material included herein is approved for public release, distribution unlimited. Not protected by copyright laws.

THIS REPORT HAS BEEN REVIEWED AND IS APPROVED

[illegible]
Glenn E. Racine, Chief
Computer and Information
Systems Division

James Gantt
James D. Gantt, Ph.D.
Director
AIRMICS

SAMeDL.TR.10.15 Sep 92

SAMeDL Pilot Project

TECHNICAL REPORT

Prepared for

U. S. Army Institute for Research in Management Information, Communication and Computer Science (AIRMICS)
115 O'Keefe Building
Atlanta, GA 30332-0800

Contract No. DAKF11-91-C-0035
CDRL A003

Prepared by

STATISTICA, Inc.
12200 Sunrise Valley Drive, Suite 300
Reston, Virginia 22091

TABLE OF CONTENTS

Figures

1.0 INTRODUCTION

The SQL Ada Module Description Language (SAMeDL) Pilot Project is an Ada Technology Insertion Program (ATIP) that explores the merits of the SQL Ada Module Extensions (SAME) methodology. SAME methodology entails interfacing Ada applications with relational Database Management Systems (DBMSs) that use SQL. Developed by Dr. Marc Graham at the Software Engineering Institute (SEI), SAMeDL is the language created to implement the SAME architecture.

1.1 *Scope*

The SAMeDL Pilot Project analyzes the value of the SAME methodology for use on future Department of Defense (DoD) DBMS applications. As a basis for the study, this project uses the SAMeDL to redesign the SQL interface layer of an existing Ada application. This effort is referred to as a "Shadow Task" because development does not impact or influence the original Ada application development. The Ada application chosen for the pilot project is the Standard Installation Division Personnel System - Third Release (SIDPERS-3) Demonstration Prototype, a large Army Management Information System (MIS). The prototype was originally designed using another SQL binding.

STATISTICA approaches the pilot project in two stages: First, the SQL binding layer is replaced with a duplicate layer implemented in SAMeDL. The Ada application design and code remains unchanged. Second, the Ada application and SQL interface layer will be redesigned to incorporate features of Ada not incorporated in the original application design due to the SQL binding. SAMeDL provides the means to use Ada features such as strong typing and exception handling.

As a byproduct of the SAMeDL Pilot Project analysis, a SAMeDL toolset is being developed for four commercial DBMSs. Intermetrics, Inc. is supporting STATISTICA in this contract by developing the support toolset. The toolset includes a SAMeDL compiler and Module Manager. The Module Manager is a library manager that maintains source code controls and performs consistency checks on SAMeDL source code and its corresponding Ada/SQL interface. The resulting toolset provides the Ada community with a more mature SAMeDL compiler that is standard across DBMSs. With each retargeting of the compiler, STATISTICA ports the Ada application layer and reports on the portability of the compiler.

1.2 Background

1.2.1 SAMeDL

In 1986, the American National Standards Institute (ANSI) issued a standard for SQL. At that time, the standard included annexes describing interfaces between SQL and programming languages, such as COBOL, FORTRAN, and Pascal. In 1987, the Ada Joint Program Office (AJPO) tasked SEI to define an Ada SQL interface. SAME is the result of SEI's efforts. The SAME uses Ada features to provide the following services to Ada SQL applications:

- A robust treatment of SQL data within application programs which effectively prevents use of null values as though they were not null whilc requiring no run time conversion of non-null data.
- A treatment of DBMS errors and exceptional conditions which is flexible, allowing application designers to decide which conditions are expected and which are irrecoverable, yet prevents any such DBMS condition from being "missed" by the application.
- An extended database description using abstract, application oriented types and the application of a strong typing discipline to SQL statements. [3]

1.2.2 SIDPERS-3

As the prime contractor, STATISTICA is developing SIDPERS-3 for the Army. This Standard Army Management Information System (STAMIS) automates applications in 36 personnel work categories.

In May 1991, STATISTICA presented a SIDPERS-3 Demonstration Prototype to the Army that featured the functional requirements described in several of the personnel work categories. The prototype validates many of the technical characteristics associated with an MIS development in Ada, such as appropriateness of Ada binding, portability of applications, and robustness of the Ada Programming Support Environment (APSE).

Although developers on the SIDPERS-3 Team have had many successes in their use of Ada, binding Ada to DBMSs has presented significant challenges. The SIDPERS-3 Team desires a binding that is straight forward to implement, insulates the application software from the specificity of a particular database implementation, and allows the Ada software engineer to use the full power of Ada. SEI's work in developing the SAME methodology has been closely followed. Early in the program, the SIDPERS-3 Team reached a consensus that SAMeDL, with an appropriate SAMeDL support toolset, would provide the necessary Ada/DBMS binding layer. However, SAMeDL lacked support tools, realistic application examples, and usage metrics (SAMeDL has not been employed as the Ada/SQL binding on any large Ada MIS program to date). Because of these barriers, the SIDPERS-3 Team felt that it was too great a technical risk to rely on SAMeDL. Hence, the SIDPERS-3 Team decided to forego the use of SAMeDL in favor of other Ada/SQL interfacing methods.

1.2.3 Intermetrics

Intermetrics has been actively involved with the SAME Design Committee since early 1988. To assist with the development of the SAME methodology and the emerging standard SAMeDL during this period, Intermetrics developed early prototype SAMeDL compilers to promote the insertion of SAMeDL into mainstream Ada applications. While these activities were useful in exhibiting proof of concept, the seed that was planted did not grow as hoped; broad acceptance and use of SAMeDL by the Ada community has not yet materialized.

A number of factors contribute to the Ada community's reluctance to insert SAMeDL into large Ada applications; STATISTICA's experience indicates that two of the most significant factors are:

1. The lack of realistic SAMeDL applications, user experiences, and related measurements.

2. A shortage of robust SAMeDL support tools.

1.3 Technical Report Overview

The purpose of this technical report is to report quarterly the results of STATISTICA's analysis and the progress of the toolset development. The report will be accumulative in that sections will be completed as activities are reported. The complete outline of the report is provided as a work plan. As subtasks are completed, the corresponding subsections will be completed.

Section 1 provides the introduction and background needed by the casual reader for understanding of the remaining sections. The project comprises two major tasks: the Shadow Task and Toolset Development. Section 2 discusses the Shadow Task during which developers migrate the existing interface layer to SAMeDL and then redesign and re-implement the application layer to incorporate SAMeDL features. Section 3 reports the efforts to develop the Module Manager, upgrade the SAMeDL compiler, and retarget the compiler to the four DBMSs.

Appendices provide supplemental information, and will be provided as the information is gathered.

2.0 TASK 1 - SHADOW TASK

To show the feasibility and benefits of using SAMeDL, STATISTICA will redesign and re-implement portions of the SIDPERS-3 prototype demonstrated to the Army in May 1991. Table 2-1 lists the Computer Software Units (CSUs) of the prototype that will compose the application layer of the SAMeDL Project. These CSUs were chosen based on the functionality and services required of the database. Each CSU in the application layer represents a complete thread of control to facilitate the analysis and measurement of performance. The prototype currently uses XDB, a Commercial Off the Shelf (COTS) SQL DBMS.

CSU Title	Description	Services	Approx. LOC*
Sgt Ssg Promotion Eligibility Unit	This CSU initially calculates a soldier's promotion points and generates PCN AAA-209, DA Form 3355-E, Promotion Point Worksheet.	Select Update Insert Fetch	2,866
Promotion Standing List Removal Action	This CSU removes a soldier from the E5-E6 Promotion Standing List and generates PCN AAA-034, a Removal From Local Recommended List memorandum.	Select Delete	2,150
* Lines of Code			

Table 2-1 CSUs in the Ada Application Layer

2.1 Existing Application Migration

Figure 2-1 shows that developers used a layered approach in the design of the prototype to isolate the database. Figure 2-2 shows the layers in more detail, with each box representing a group of Ada packages. Table 2-2 depicts the correlation between the layers in Figure 2-1 and the Ada packages depicted in Figure 2-2.

Design decisions made early in the project were premised on using a commercial DBMS. Inherently, SQL databases, limited to the standard SQL data types, do not support strong data typing. To avoid anonymous types and to add reliability and maintainability to the system, the SIDPERS-3 Team created a layer of Type packages. These packages declare subtypes corresponding to each column in the database. In the Database Support layer, data retrieved from the database is explicitly converted to SIDPERS-3 types before processing by the application layer. The Database Support layer isolates the SIDPERS-3 application from changes that may occur to the database.

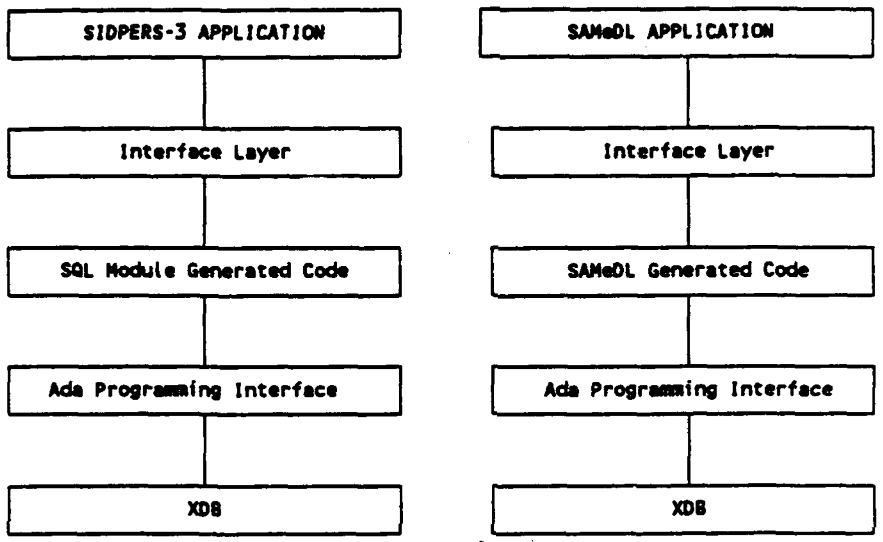


Figure 2-1 Layered Approach

The SIDPERS-3 Team developed a Man Machine Interface (MMI) to handle all user interfaces, including reports. The data is passed between the MMI and the application as string objects to support all of the SIDPERS-3 data requirements. The Types packages written to support the database interface also serve the MMI by eliminating a conversion layer between the application and the MMI.

Finally, to avoid the bulky processing required to test for missing data (null values), the SIDPERS-3 Team created the database with NOT NULL columns.

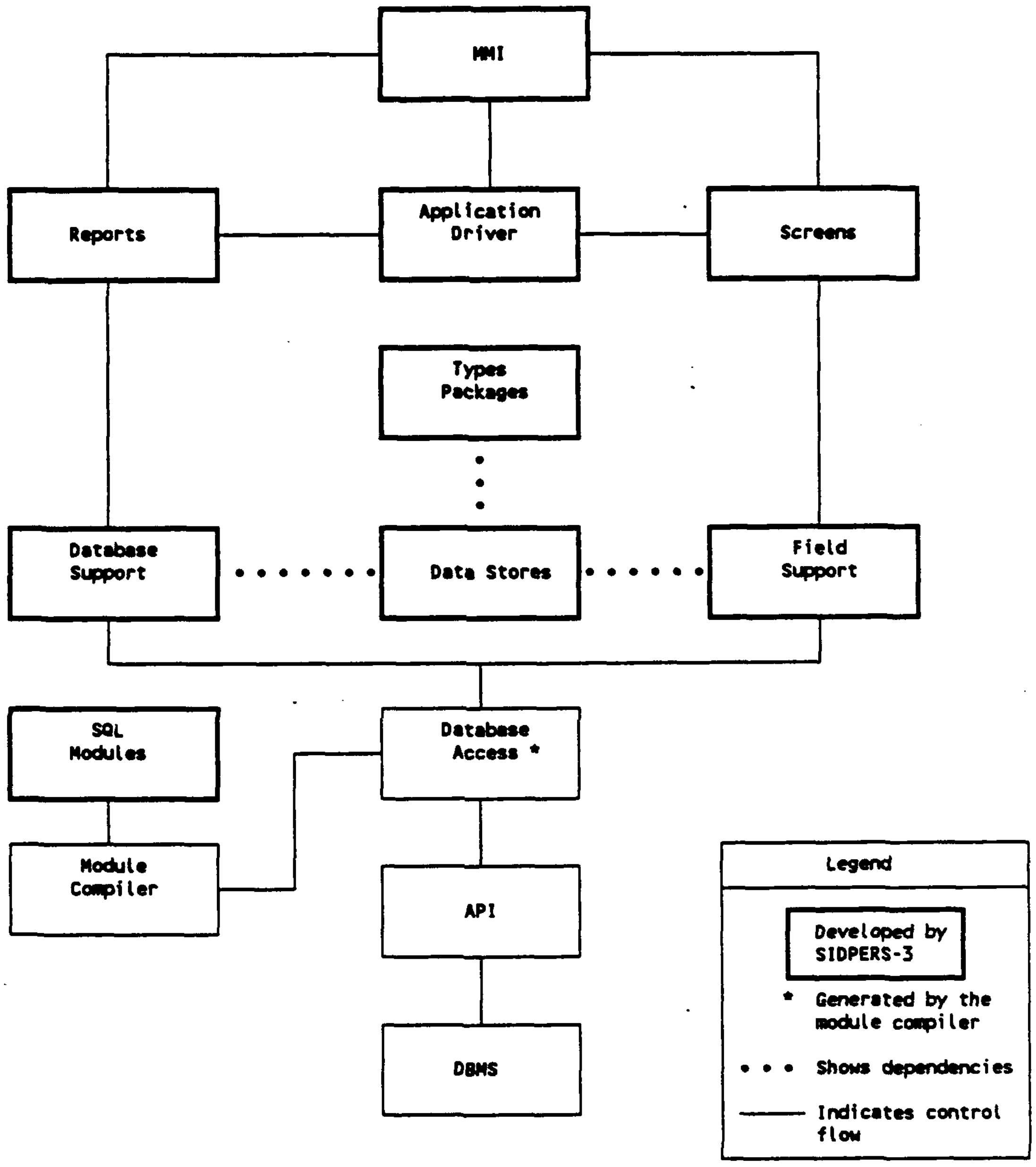


Figure 2-2 The SIDPERS-3 Prototype Architecture

Figure 2-1 Layer	Figure 2-2 Box	Conversion Efforts
Ada Application	Application Driver Reports Screens Field Support Data Stores Types Packages	No changes will be made to the application layer.
Interface Layer	Database Support	Remove Commit/Rollback procedures; add functions to convert SAMeDL types to SIDPERS-3 types; remove exception handlers.
SQL Module Generated Code	Database Access	Replace SQL modules with SAMeDL modules.

Table 2-2 SIDPERS-3 Layers/Architecture Correlation

There are no changes to the Ada application layer in the Application Migration subtask. The prototype uses XDB's module compiler as the SQL binding to the Ada application layer. For the SAMeDL Pilot Project, this layer is replaced with a SAMeDL layer. Subsection 2.1.1 discusses this conversion process. The Database Support packages require modifications to remove features provided by SAMeDL and add conversions of SAMeDL types to SIDPERS-3 types (declared in the Types packages). The changes required to the interface layer are discussed in Subsection 2.1.2.

2.1.1 Module Input Migration

The migration of the SQL input modules involves replacing the XDB SQL Modules with SAMeDL Modules.

2.1.1.1 Approach

The project team quickly completed this step by referring to the SQL statements in SQL module files for the specific SELECT, UPDATE, DELETE, and INSERT statements. Also, since only certain tables and columns are required by the chosen CSUs, the SQL modules provided the needed information for declaring the domain types and tables in the SAMeDL Definition and Schema modules. The SAMeDL modules were submitted to the SAMeDL compiler for generation of the Ada packages. For the XDB DBMS, the Ada packages generated by the SAMeDL compiler replace the Database Access files (refer to Figure 2-2) in the SIDPERS-3 model.

2.1.1.2 Observations

The ease of the SQL to SAMeDL module conversion can be attributed to the module binding provided by XDB and the layering approach used by STATISTICA to isolate the database. Had the SIDPERS-3 Team chosen to use an embedded SQL interface without isolating the database, the application layer would have required major changes to migrate to SAMeDL.

2.1.2 Interface Migration

The interface migration involves modifying the interface layer (refer to Figure 2-1) or Database Support packages (refer to Figure 2-2). The Database Support packages are changed to reference the SAMeDL modules rather than the Database Access packages generated by the XDB SQL module compiler.

2.1.2.1 Approach

The interface layer represents a set of Ada packages that provides the conversion of SQL types to user-defined types specific to the SIDPERS-3 application. The Database Support packages directly call the Ada packages generated by the XDB SQL module compiler. The Database Support packages were modified to interface with the Ada code generated by the SAMeDL compiler. These packages provide database control functions such as rollback, commit, open database, close database, and error handling. Since these functions are provided in the SAMeDL packages, the functions were removed from the Database Support packages.

The SIDPERS-3 MMI uses one database table for processing error messages. A SAMeDL module was written to provide this service, and the appropriate MMI package was modified and recompiled to interface with the new SAMeDL module.

2.1.2.2 Observations

The migration to SAMeDL required no modifications to the application layer. Major changes, however, were required to the Database Support packages to interface with the SAMeDL modules.

There is a conflict in the manner in which the SAMeDL modules and the MMI display error messages. The SAMeDL error handling procedure uses the Text_IO package to output to the screen a message containing the error code. In contrast, the MMI provides a standard display of all user messages. To take advantage of the existing MMI procedure, the SAMeDL error handling procedure could be modified to call the MMI to display the error message. This solution, however, creates a circular dependency between the SAMeDL layer and the MMI, as shown in Figure 2-3.

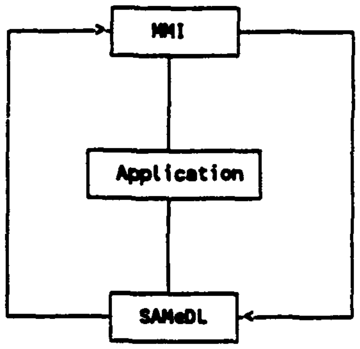


Figure 2-3 The Circular Dependency between SAMeDL and MMI

The solution to the circular dependency problem is the creation of a separate error handling package, as shown in Figure 2-4. Upon return of the error code, negative SQLCODE, from the database, the SAMeDL layer calls the error handler in the Database Errors package. The error handler uses the function to display user message provided by the MMI.

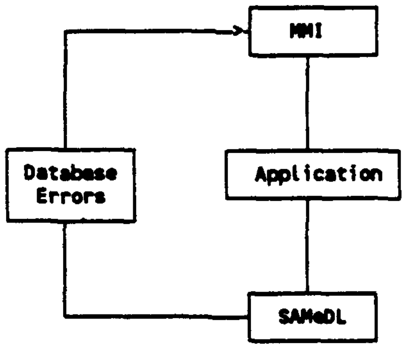


Figure 2-4 The Circular Dependency Solution

2.1.3 Data Migration

The database for the SIDPERS-3 Demonstration Prototype contains approximately 200 tables. Only 34 of these tables are accessed by the two CSUs in the SAMeDL Pilot Project. Data migration involves the transfer of the 34 tables to a smaller XDB database. Once the subset database is created, it can be used for creating and loading databases in Informix, Oracle, and Sybase.

2.1.3.1 Approach

STATISTICA found most of the table names for the two CSUs in the SQL modules that were originally written for the SIDPERS-3 Demonstration Prototype. Other tables names, used as lookup tables or pop-up windows, were found by perusing the Database Access packages.

An SQL procedure was written using the XDB system tables to generate the script file automatically to create the tables in a new XDB database. Another script file was written to create indexes identical to the original SIDPERS-3 indexes. The two script files contain standard SQL statements; therefore, they can be used to create the tables and indexes in the other DBMSs. Once the tables and indexes were created, the export and import utilities provided by XDB were used to move the data to the new database.

The new database was tested for completeness using the original SIDPERS-3 Demonstration Prototype. During testing, it was discovered that the MMI uses a database table to display user messages. This table was added to the new database.

2.1.3.2 Observations

The primary reason for creating a subset of the SIDPERS-3 database was to facilitate loading data to Informix, Oracle, and Sybase. The conversion of the database to these DBMSs will be described in Subsection 2.2.3.

2.1.4 Test

Testing in the Application Migration subtask focuses on answering two questions:

1. Is the application correctly converted, retaining all functionality of the original application?

2. Are there differences in performance between the original application and the new SAMeDL application?

2.1.4.1 Test Plan

To test the correctness of the SAMeDL application, the project team is using the test cases created by the SIDPERS-3 team for the SIDPERS-3 Demonstration Prototype. The CSU Test Cases are included in this report as Appendix G. Using the test data in the XDB database, the project team will execute the test procedures on both the original application and the SAMeDl application. Any discrepancies in the SAMeDL application will be noted, corrected, and retested until the SAMeDL application is functionally equal to the original, baseline application.

Performance differences between the original application and the SAMeDL application are measured by recording system clock time at identical points within each application. The clock time is captured at the entry/exit to certain procedures and before/after calls to the database. The process time for a procedure or database call is calculated as the exit clock time minus the entry clock time. Each selected procedure is tested repeatedly, capturing the process time so that the minimum, maximum and average processing times can be calculated and recorded for each procedure or database call.

2.1.4.2 Test Results

The project team executed the test procedures contained in the CSU Test Cases, Appendix G, against the original SIDPERS-3 Demonstration Prototype and the converted SAMeDL application. Each test case was performed with duplicate results from each application. The SIDPERS-3 Demonstration Prototype had been correctly converted to SAMeDL without loss of functionality.

The project team is experiencing difficulty in measuring performance differences between the two applications. The smallest measure of time returned by the Interactive UNIX operating system is tenths of seconds. This time increment is not granular enough to show differences in processing time.

One solution is to run a "counting" process in the background while the Ada application runs in the foreground. However, this approach failed due to a conflict between the Ada run time and the UNIX process scheduler. With Alsys, the Ada run time executes as a separate process on top of the operating system. The conflict is created when the Ada tasks are scheduled by the Ada run time independently of the UNIX process scheduler. Since performance could be a key discriminator for SAMeDL, the project team will continue to explore other methods to measure processing time.

2.2 New Application Development

For the New Application Development subtask, STATISTICA will use the same CSUs listed in Table 2-1. This approach provides a baseline from which comparisons between methodologies can be made as to design without functionality differences in the application layer. In the New Applicaticn Development subtask, STATISTICA will redesign and re-implement some of the Ada application layer to analyze the SAMeDL features and to assess the value added using the SAME methodology.

Further study of the SIDPERS-3 Demonstration Prototype design, as depicted in Figure 2-2, reveals the following points:

1. The Screens box represents a series of Ada packages, the design of which is dictated by the MMI. The control flow within the each package is determined by the active screen name and the key used to exit that screen. Although tailored for each CSU, these packages rely heavily on the MMI and can be considered part of the MMI. Since a user interface is usually considered external to, and separate from, the application, the packages were not modified to include SAMeDL.

2. The Screens packages are dependent on Field Support packages for validating user-entered data, retrieving data from the database for pop-up or help windows, and storing data in a Data Store for use during later processing.

3. The Data Stores box represents several package specifications that declare objects for temporary storage of values either retrieved from the database or entered by the user. The data is stored until needed for reports, validation or calculations. The objects in the Data Stores are declared as user-defined subtypes found in the Types packages.

4. The Reports box is also dependent on the MMI for report utilities; however, the packages in this group contain procedure calls to retrieve and update data in the database. The package that calls MMI report utilities for formatting the CSU specific report was not unchanged.

5. The Database Support box represents an interface layer to the Database Access packages generated by the XDB module compiler. These packages were modified extensively in the previous subtask (refer to Section 2.1.2) and, therefore, require only minor changes.

2.2.1 Approach

The project team used the following approach to redesign the two CSUs:

1. Analyze the data requirements to identify objects or specific data groups. For example, data identifying a soldier, such as SSN and name, compose a Soldier object. The data identifying a Unit (i.e. UIC and name) are attributes of the Unit object.

2. Build SAMeDL domains and support packages around each object. For each object, a SAMeDL definition module is written to declare the attribute domain of the object. A corresponding SAMeDL abstract module is written to provide all database operations required to use the object. An example of the definition and abstract modules written for the Unit object is shown in Figure 2.5.

3. Modify the database schema to represent the real world. Where appropriate, the project team changed database columns to allow null values. The project team then created a SAMeDL schema module to match the new database schema. An example of the schema module is shown in Figure 2.6 on page 16.

4. Compile SAMeDL modules. For each definition module, the SAMeDL compiler generates an Ada specification package (e.g. Unit_Def) in which derived types are declared for each domain. Additionally, the SAMeDL compiler creates a set of Ada packages (e.g. Unit_Abs specification and body) for each abstract module. This set of Ada packages is the abstract interface defined in the SAME architecture. In the XDB version of the SAMeDL compiler, a third set of packages (actually generated by the XDB module compiler) is required to implement the database calls in the abstract module. This third set of packages becomes the concrete interface in the SAME architecture.

5. Redesign and modify application layer. The project team identified several goals in redesigning the application.

   a. Replace weaker subtypes in the Types package with SAMeDL derived types. The objectives of this goal are to remove the redundant type declarations and enforce compiler time checks through derived limited private types.

   b. Hide implementation details of the Data Stores by

```
definition module UNIT_DEF is

  domain UIC_Domain is new SQL_CHAR (Length = > 6);
  domain UNAME_Domain is new SQL_CHAR (Length = > 60);
  domain UNIT_ITEM_COUNT_Domain is new SQL_INT;

  exception Record_Not_Found;

  record UNIT_INFO_RECORD is
    UIC   : UIC_Domain;
    UNAME : UNAME_Domain;
  end UNIT_INFO_RECORD;

end UNIT_DEF;


--!reference UNIT_DEF
--!reference REMOVAL_DEF
--!reference DEMO
with UNIT_DEF; use UNIT_DEF;
with REMOVAL_DEF;
abstract module UNIT_ABS is
 authorization DEMO

 record COUNT_RECORD is
   Number : UNIT_ITEM_COUNT_Domain;
 end;

 procedure Count_Units is

   select UNIT_ITEM_COUNT_Domain(count(*))
     into  Number_of_Items  : COUNT_RECORD
   from UNIT;

 cursor UIC_Cursor for

   select UIC,
        UNAME
   from UNIT
   order by UNIT.UIC;


 procedure Get_Attached_Unit_Info
     (With_SSN : Removal_Def.SSN_Domain) is

   select UNIT.UIC,
        UNAME
   into UNIT_INFO : UNIT_INFO_RECORD
   from ATTACHMENT, UNIT
   where (ATTACHMENT.SSN = With_Ssn and
        ATTACHMENT.RSN_ATCH = 'A') and
        UNIT.UIC = ATTACHMENT.UIC;


 procedure Get_Current_Unit_Info
         (With_SSN  :  Removal_Def.SSN_Domain) is

   select UNIT.UIC,
        UNAME
   into UNIT_INFO : UNIT_INFO_RECORD
   from CURR_ASSIGNMENT, UNIT
   where CURR_ASSIGNMENT.SSN = With_Ssn and
      UNIT.UIC = CURR_ASSIGNMENT.UIC;

end UNIT_ABS;
```

Figure 2-5 Definition and Abstract Modules for Unit Object

encapsulating Data Stores within support packages. The objective of this goal is to utilize information hiding and to give the design a more "object-oriented" flavor.

c. Retain strong data typing up to the point where the MMI controls the data. The objective of this goal is to process the data using the operations defined for each data type and take advantage of compiler-time, rather than run-time, error detection.

The resulting SAMeDL redesign architecture is shown in Figure 2-7.

A comparison of the SIDPERS architecture (refer to Figure 2-2) and the redesign architecture (shown in Figure 2-7) indicates how the first two goals of the redesign were met. With the Derived Types packages generated by the SAMeDL compiler, the previous Types

```
--!reference UNIT_DEF
--!reference REMOVAL_DEF
with UNIT_DEF;
with REMOVAL_DEF;
schema module DEMO is

  table SOLDIER is
    SSN    not null          : REMOVAL_DEF.SSN_Domain,
    NAME_IND                 : REMOVAL_DEF.NAME_IND_Domain,
    PERM_GR_AD               : REMOVAL_DEF.PERM_GR_AD_Domain
  end SOLDIER;

  table CURR_ASSIGNMENT is
    SSN    not null          : REMOVAL_DEF.SSN_Domain,
    UIC                      : UNIT_DEF.UIC_Domain
  end CURR_ASSIGNMENT;

  table ATTACHMENT is
    SSN    not null          : REMOVAL_DEF.SSN_Domain,
    UIC                      : UNIT_DEF.UIC_Domain,
    RSN_ATCH                 : REMOVAL_DEF.RSN_ATCH_Domain
  end ATTACHMENT;

  table UNIT is
    UIC    not null          : UNIT_DEF.UIC_Domain,
    UNAME                    : UNIT_DEF.UNAME_Domain
  end UNIT;

end DEMO;
```

Figure 2-6 Sample Schema Module for Redesigned Application

packages becomes redundant. The Data Store packages are replaced with objects of SAMeDL derived types declared within the package bodies of the Field Support packages. Access to the Data Stores is gained through services (i.e. Get_Object, Put_Object) already provided by the Field Support procedures. These procedures are modified to reference the local objects rather than the objects declared in external package specifications. The other functionality provided by the Field Support packages is retained so that the Screens packages require no modifications.

The project team modified the Database Support package to reference the procedures in the Abstract Interface (generated from the abstract modules) rather than the Database Access packages.

The third goal is not as easy to attain. The project team determined that the point at which the MMI controls the data is just prior to calling the Field Support procedures and prior to calling report utilities in Reports packages. When the Screens packages call the Field Support procedures the data value is passed as a string type. Conversion to derived types is handled by the Field Support procedures, thus hiding the conversion details and

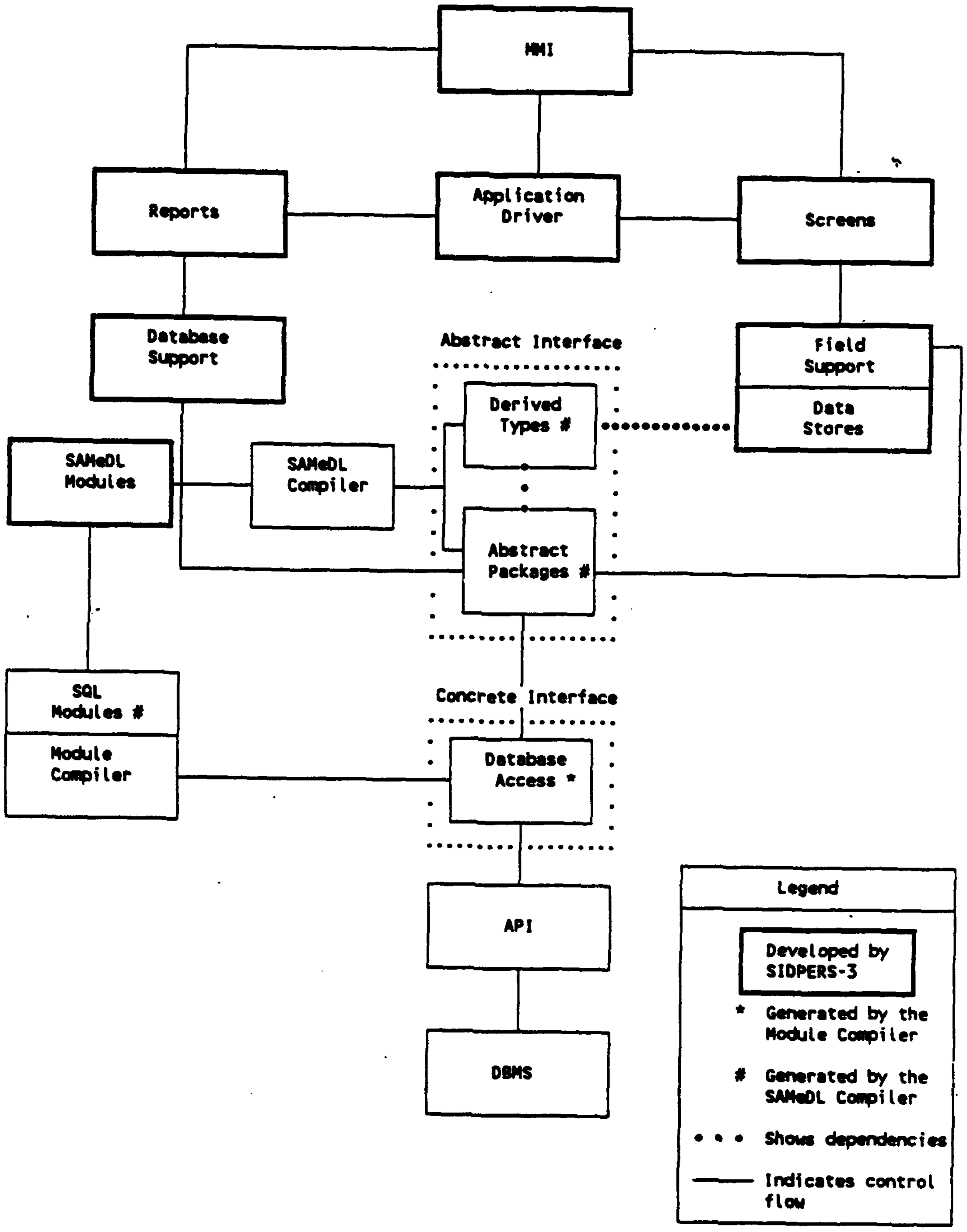


Figure 2-7 The SAMeDL Redesign Architecture

isolating the calling packages from any future changes to data types.

2.2.2 Observations

SAMeDL must be the basis for design. Once the database design has become stable, the SAMeDL modules can be designed and written. Database stability is the key to the success of implementing SAMeDL. Because the application layer is built on top of the SAMeDL data types, any change in the database schema resulting in a change to SAMeDL types requires modifications to and recompilation of the application layer.

For example, changing a not null column to a null-bearing column in the database would require a modification to the corresponding domain in the SAMeDL definition module. The new definition module would be recompiled through the SAMeDL compiler, generating a new Ada specification. The data type generated would become a limited private type, and some operations (i.e. the Ada ":=" operator) on the not null type would become invalid.

One solution to this problem is to provide an additional layer of abstraction between the application and the SAMeDL abstract interface. In our redesign, the Database and Field Support packages, in essence, provided this layer. However, the cost of this solution is additional processing and response time.

The SAME architecture supports an object-based design. The data requirements of the application were analyzed to identify object classes or data types that could be grouped into packages. In addition to conversion functions, Field Support packages were modified to hide the objects (designated as Data Stores) needed for temporary storage of data.

SAMeDL is as complex to use as Ada. It is a hybrid of Ada and SQL, offering the best features of Ada and allowing the user to specify database services in SQL-like statements. The user must, therefore, be proficient in both Ada and SQL, while learning a third programming medium. Program managers must consider training or learning curve impacts on the development of SAMeDL applications.

2.2.2.1 Strong Typing

The success of retaining the SAMeDL strong data typing in the application layer depends on how tightly interleaved the user interface is with the application design. The design of the SIDPERS-3 application is dictated by the SIDPERS-3 MMI. As seen in the first stage of this pilot project, inserting SAMeDL without modifications to the SIDPERS-3 application proved to be of no worth. This was largely due to the interface design where data is passed as string objects. In the redesign (second stage of the project), the application layer was modified to use the SAMeDL derived types. Functions to convert SAMeDL types to string types were created in the Database and Field Support packages to accommodate the MMI.

2.2.2.2 Error Handling

Any SQL statement executed by the database has the potential for failure. Frequently, an application is designed to catch the predictable errors (e.g. no record found) and forgets to check for the unpredictable, unrecoverable failures (e.g. disk error). The SAME methodology handles unexpected errors, while providing a flexible treatment of database errors that allows the application to define errors that are acceptable and expected.

The application programmer defines the database errors that are tolerable in the definition modules by declaring a status map, as shown in Figure 2-8.

```
exception Data_Definition_Does_Not_Exit;
exception   Insufficient_Privilege;

enumeration Operation_Status is (Disk_Error, Data_Conversion_Error,
                                 Invalid_SQL_Statement, Not_Found,
                                 Okay);

status Operation_Map named Result_Of_Operation
          uses Operation_Status is (
          -600 .. -699                 =>        Disk_Error,
          -500 .. -599                 =>        Data_Conversion_Error,
          -300 .. -499                 =>        Invalid_SQL_Statement,
          -101, -110, -113             =>        raise Data_Definition_Does_Not_Exit,
          -25                          =>        raise Insufficient_Privilege,
          0                            =>        Okay,
          100                          =>        Not_Found);
```

Figure 2-8 Sample Status Map Declaration

As seen in Figure 2-8, the declaration of the status map may include **raise** statements that raise exception handlers for those errors that are unpredictable or unrecoverable. Expected errors are mapped to members of the enumeration type, Operation_Status.

The negative integers are values of the ANSI standard variable, SQLCODE. The status map provides a direct correlation between the returned SQLCODE and application-defined error conditions. It should be noted that the standard specifies only two return values: 0 for "success" and 100 for "row not found." Any other SQLCODE must be a negative integer and is implementation-defined. In other words, the negative integers differ between DBMSs, and declaration of the status map should be isolated for portability reasons.

An application is not required to declare a status map. In this case, upon return of a database error (negative SQLCODE), the procedure Process_Database_Error, declared in package SQL_Database_Error_Pkg, is called to raise the exception SQL_Database_Error. The application should provide an error recovery routine for handling the SQL_Database_Error exception.

2.2.2.3 Null Handling

Declaring domains as null-bearing increased the complexity of programming the application code, thus decreasing the productivity of the application programmer. The SAME support packages (e.g. SQL_Char_Pkg) provided the operations necessary for the limited private types; however, the application programmer should be trained on the effective use of the support packages. The restrictiveness of the limited data type can be circumvented; however, the reliability and maintainability of the program is lost.

2.2.3 Multiple Target Databases

Targeting the application across databases required minimal effort and changes. There are three areas that required modifications:

1. CONNECT statements;
2. Database status map; and
3. The transfer of data from one database to another.

The first area involves modifications to CONNECT statements. As shown in Figure 2-9, the CONNECT statement is slightly different across the four databases.

The second area is modifications to the status map declared in the definition module to handle database errors. As discussed in Section 2.2.2.2, the status map must be modified to reference the new DBMS codes, if an application chooses to map application-defined errors directly to DBMS-specific return codes .

XDB

```
CONNECT;
```

Oracle

```
CONNECT User_ID Password [USING Database_Name];
```

Informix

```
CONNECT Database_Name;
```

Sybase

```
CONNECT SERVER Server_Name;
CONNECT Database_Name;
```

Figure 2-9 CONNECT Statements

The third area is transferring data from one database to another. This process involves the following steps:

1. Create the database and table using either SQL scripts or the appropriate user interface.

2. Export the data from the source database to a format acceptable to the target database.

3. Import or load the data into the target database.

The project team created script files that contain standard data definition language for creating the database, creating tables and indexes, and granting the correct privileges to user accounts. Both the Informix and Oracle database provide utilities for loading data from ASCII files. The use of script files and DBMS-supplied utilities made the loading of the Informix and Oracles database easy.

3.0 TASK 2 - TOOLSET DEVELOPMENT

The purpose of this section is to document the technical activities associated with the SAMeDL Toolset Development task. This section is organized as follows:

1. Section 3.1, SAMeDL Module Manager Development, covers the SAMeDL Module Manager development effort.

2. Section 3.2, SAMeDL Compiler Upgrade, documents the work associated with upgrading the existing Intermetrics SAMeDL compiler to support the most current definition of SAMeDL.

3. Section 3.3, SAMeDL Compiler Retarget, reports on the technical activities associated with retargeting the Intermetrics SAMeDL compiler backend to the four supported DBMSs.

4. Section 3.4, Other Support Tool Development, covers the effort associated with developing additional tools, if any, other than the basic compiler and the Module Manager.

3.1 SAMeDL Module Manager Development

3.1.1 Design

Intermetrics successfully performed initial prototyping work on the Sun4 as a feasibility study for the SAMeDL Module Manager design. The top level design document is incorporated into this technical report as Appendix B. The objective of the Module Manager is to provide the user with reasonable management of the written SAMeDL modules and the Ada interfaces generated by SAMeDL. The Module Manager implementation will be simple, and as portable as possible.

From the user's point of view, the interface is line oriented (like Verdix). Functionality includes library creation/ deletion, SAMeDL information listings (i.e., time/date of compilation, dependencies, associated host files for input source code and generated interfaces), and Ada compilation ordering information for the generated Ada interfaces. A programming interface has been developed for use by the SAMeDL compiler to aid in separate compilation and information retrieval/generation from/to the library.

Time and resources permitting, Intermetrics will add "nice to have" features discovered during testing. These features or modifications fall under two categories:

1. Information Presentation. The way library information is presented to the user may be enhanced.

2. Convenience. Intermetrics may add functions that automate user activities based on information already managed and available. For example, archives could be generated from the created object files (where concrete interfaces take the form of C/ESQl). Additionally, shell scripts could be created that automate the compilation of the generated Ada packages.

3.1.2 Code

Intermetrics has developed and integrated the Module Manager into the SAMeDL compiler. The foundation for this work is heavily based on the prototype of the Module Manager developed by Intermetrics during the design phase. The Module Manager is written in Ada and developed on a Sun4 workstation using the Verdix Ada Development System (VADS). Following initial testing on the Sun4, the Module Manager has been successfully ported to the 386 computer under Interactive UNIX and the Alsys compiler.

3.1.3 Test

Intermetrics performed initial testing of the Module Manager on both the Sun4 and the 386 computer. With respect to the compiler interface, SAMeDL modules with miscellaneous interdependencies were successfully processed by the compiler. The related information generated by the Module Manager for the library was then manually examined, either through the debugger or by the Module Manager user interface. To test the user interface functions, SAMeDL modules and related generated interfaces were entered into a SAMeDL library. The user commands were successfully tested using various combinations of options and parameters.

Informal testing will continue on the Module Manager as part of the development and testing of the SAMeDL compiler.

3.2 SAMeDL Compiler Upgrade

3.2.1 Design

The SAMeDL compilers were developed by Intermetrics from an existing compiler. The original compiler was targeted to the October 1991 version of SAMeDL. The SAMeDL compilers for the Pilot Project are targeted to the November 1991 version of the language, developed primarily at SEI.

An incremental approach was taken to upgrade the SAMeDL compiler from the October to the November version of the language. The additional features were partitioned into three logical sets. The first set of upgrades was implemented for the delivery of the Informix SAMeDL compiler. The second compiler was implemented for XDB and included both the first and second sets of upgrades. The Oracle and Sybase SAMeDL compilers implemented the full November 1991 version of the language.

The incremental approach to upgrading the SAMeDL compiler was advantageous to both the Shadow Task and the toolset development task of the Pilot Project. From the point of view of the Shadow Task, the incremental upgrades meant that the first SAMeDL SDE delivery could be made soon after contract award, enabling STATISTICA to start using the SAMeDL toolset as early as possible. By maximizing the time STATISTICA had to use the toolsets, the feedback to Intermetrics was integrated into subsequent compilers, resulting in a better set of SDEs for the final delivery.

3.2.2 Code

Coding of the SAMeDL compiler front-end upgrades is performed on a Sun4 with a Verdix Ada compiler. Once the upgraded code is tested on this development platform, the improved compiler is ported and tested on the delivery platform. Compiler back-end improvements and DBMS retargeting is performed on the Sun4 development platform and then ported to the delivery platform to test using the target DBMSs. Use of the Sun4 development platform and the PC-386 delivery platform in this way enables both front-end and back-end upgrades to be performed simultaneously.

All source code, including the SAMeDL compiler source code, the Module Manager source code, and the SAMeDL standard packages is maintained in a central repository on the Sun4 platform under strict configuration management policies. At delivery time, the configuration management system registers a release of the current code, which is used to build the delivery executables.

3.2.3 Test

Acceptance testing of the SAMeDL compiler is based on the SAMeDL Development Environment Test Plan and Intermetrics' version of the SAMeDL Language Reference Manual (ILRM). The Test Plan is incorporated in the technical report as Appendix C. The ILRM is incorporated as Appendix D. The Test Plan contains procedures for testing the Module Manager and source code for the compiler test suite. A cross reference of test procedures to ILRM sections is provided in Chapter 4 of the Test Plan.

Testing of the SAMeDL compiler is divided into three basic types:

1. Correct Tests. This set of test procedures verifies that the SAMeDL compiler recognizes and processes proper syntactical and semantic constructs. Proper syntactical and semantic constructs are defined by the ILRM.

2. End-To-End Tests. This set of test procedures verifies that the output of the SAMeDL compiler functionally (as defined by the ILRM) interfaces with the target database.

3. Error Tests. This set of test procedures verifies that the SAMeDL compiler identifies improper syntactical and semantic constructs (as defined by the ILRM) as errors.

3.3 SAMeDL Compiler Retargets

Once upgraded, the SAMeDL compiler will be retargeted to four DBMSs: Informix, XDB, Oracle, and Sybase. To prioritize the compiler retargets, Intermetrics:

1. Identified commonality across the programming interfaces provided by each DBMS vendor. Informix, Oracle, and Sybase have a standard C with embedded SQL (C/ESQL) programming interface. XDB has an Ada/SQL module language interface, and does not provide the C/ESQL. Since C/ESQL is the interfacing technique currently used in Intermetrics compiler, Intermetrics will retarget one of the three DBMSs that generates C/ESQL for the concrete interface. Once this is done, the other two DBMS can be retargeted quickly.

2. Targeted the DBMSs that are most widely used to promote use and broad availability of SAMeDL across the Ada community. XDB is a product primarily used by the Army; the remaining three DBMSs are commercial products that are widely available to academia, government, and industry.

Using the above rationale, it is clear that the DBMSs can be partitioned into two distinct groups:

1. Group A - Informix, Oracle and Sybase

2. Group B - XDB.

If Intermetrics implements one DBMS from each group first, it is reasonable to assume that the SAMeDL compiler can be retargeted to all four DBMSs. Intermetrics will retarget the SAMeDL compiler for Informix first, followed by XDB, and then either Oracle or Sybase.

3.3.1 Informix

The front-end upgrade and back-end retarget of the SAMeDL compiler to the Informix DBMS proceeded smoothly through its completion and delivery on March 13, 1992 to STATISTICA. The Informix DBMS provides a stable and reasonably complete C/ESQL interface. Thus, a minimal number of extensions to SAMeDL were needed to provide a satisfactory SQL interface to Informix.

The SAMeDl compiler currently implements all of the SAMeDL capabilities described in Appendix D, SAMeDL Language Reference Manual.

3.3.2 XDB

The SAMeDL compiler targeted to the XDB/PC-386 platform was delivered to STATISTICA on May 7, 1992. Several differences between Informix and XDB contributed to making the retarget to XDB more technically challenging than the Informix retarget.

The Computer Associates XDB DBMS provides an SQL-Ada module compiler. The SQL-Ada module compiler was chosen as a back-end target for the SAMeDL compiler for XDB. The modification of the compiler back-end from the embedded C/SQL architecture to the SQL-Ada module compiler went relatively well considering the magnitude of the change.

Early in the retargeting process, Intermetrics discovered software problems in the XDB SQL-Ada module compiler that would significantly limit the functionality of the XDB SAMeDL compiler. The most severe problems were corrected by Computer Associates. Some problems involve lack of conformance to SQL and SQL module language standards, as defined by FIPS PUB 127-1. Workarounds to these conformance problems are suggested in Appendix F, SAMeDL User Manuals. When workarounds were not appropriate, Intermetrics added semantic checks to the SAMeDL compiler to warn users that certain features of the language are not adequately supported by XDB.

The XDB SAMeDL compiler implements all of the SAMeDL language described in Appendix D, SAMeDL Language Reference Manual.

3.3.3 Sybase

The Sybase SAMeDL compiler targeted to the Sun/Sparc platform was delivered on July 7, 1992 to STATISTICA. The Sybase SAMeDL compiler is targeting a hardware/software platform that is different from the platform for the Informix and XDB SAMeDL compilers. Also, the SDE for Sybase is the first delivery to support a complete front-end compatible with the November 1991 version of the SAMeDL LRM [5].

As the time to commence the Sybase retarget effort approached, Intermetrics discovered that Sybase no longer provides a C/ESQL product for the PC-386 platform. The decision to use a Sun/Sparc platform for the Sybase SDE was made for three major reasons:

1. Intermetrics has a Sun/Sparc platform at their facility on which they could do the work, and to which they could provide STATISTICA access in support of the Shadow Task.

2. Sybase provides a C/ESQL product for the Sun/Sparc platform.

3. Intermetrics felt that use of this platform would enable the contract to proceed on schedule, whereas choosing an alternative compiler architecture or DBMS might not.

The full configuration for the Sybase SDE consists of a Sun/Sparc machine running SunOS 4.1.1, the Verdix Ada Compiler Version 6.0.3c and the Sybase DBMS Version 4.8 with C/ESQL.

The Sybase retarget proceeded smoothly, with only minor problems discovered in using the Sybase DBMS. Sybase restricts the use of a cursor update statement to several qualifications, such as having a unique index on the table column to be updated. Consequently, attempting to use the cursor update statement without meeting the Sybase prerequisites will result in a Sybase error either at run-time or during the final phase of compilation by the Sybase C/ESQL precompiler. Intermetrics reported this problem to the Sybase technical support staff.

To be compatible with the new platform, some commands generated by the Module Manager had to be altered. For example, the **sde.mkscript** commands were changed to emit a script compatible with the Verdix Ada compiler. The changes to the Module Manager were minor and were documented in the Sybase SDE User Manual.

The complete front-end supports all features of SAMeDL, including user-defined base domains. Some of the new capabilities of the Sybase SAMeDL compiler differ from the SAMeDL defined in the November 1991 version of the LRM. The correct use of all SAMeDL features, as implemented by Intermetrics, is found in Appendix D. Further clarification of implementation-dependent features is found in the Sybase SDE User Manual, in Appendix F.

3.3.4 Oracle

Intermetrics delivered the SAMeDL compiler targeted to the Oracle/PC-386 platform on August 7, 1992 to STATISTICA. Intermetrics produced the Oracle SAMeDL SDE during 1 month of intense activity.

The first three deliveries, with accumulative front-end upgrades, were performed during 2-month intervals. The Oracle SDE required no front-end upgrade and was purely a retarget. The inherent portability of the SAMeDL compiler architecture enabled this retarget to be performed in one-half the time of the earlier retargets. In addition to the actual retarget, the 1-month time included updating the documentation and packaging the media.

One reason that the SDE for Oracle was built so quickly is that the Oracle DBMS has few program errors. The only SAMeDL feature not supported by Oracle is the null-bearing host variable in the SQL **where** clause. This implementation-dependent feature is documented in the Oracle SDE User Manual, in Appendix F.

3.4 Other Support Tool Development

Intermetrics recently identified two tools that may serve as an aid to the Shadow Task:

1. A SAMeDL Language Sensitive Editor (LSE).
2. A SAMeDL syntax checker.

Intermetrics implemented an early version of the LSE for the Sun4 based on the October 1990 SAMeDL Language Reference Manual. An effort will be made to upgrade and port the LSE to the 386 computer. The LSE utilizes emacs with appropriate e-lisp bindings based on the SAMeDL grammar.

The syntax checker will be incorporated into the SAMeDL compiler. Until a completed SAMeDL compiler is available, the syntax checker will help in the early development of the Shadow Task.

APPENDIX A

Acronyms and Bibliography

ACRONYMS

AIRMICS	Army Institute for Research in Management Information, Communication and Computer Science
AJPO	Ada Joint Program Office
ANSI	American National Standards Institute
APSE	Ada Programming Support Environment
ATIP	Ada Technology Insertion Program
COTS	Commercial Off The Shelf
CSU	Computer Software Unit
C/ESQL	C with embedded SQL
DBMS	Database Management System
DoD	Department of Defense
ILRM	Intermetrics' SAMeDL Language Reference Manual
LOC	Lines of Code
LRM	Language Reference Manual
LSE	Language Sensitive Editor
MIS	Management Information System
MMI	Man Machine Interface
SAME	SQL Ada Module Extensions
SAMeDL	SQL Ada Module Description Language
SEI	Software Engineering Institute
SIDPERS-3	Standard Installation Division Personnel System - Third Release
STAMIS	Standard Army Management Information System
VADS	Verdix Ada Development System

BIBLIOGRAPHY

[1] *Database Language - SQL with Integrity Enhancements*. American National Standards Institute, X3.135-1989.

[2] *Database Language - Embedded SQL*. American National Standards Institute, X3.168-1989.

[3] Graham, Marc H., "Down in the Details, Lessons Learned in Interfacing Ada and SQL." ACM Tri-Ada '90 Conference, Baltimore, MD, December 1990.

[4] Graham, Marc H., "Guidelines for the Use of the SAME." Software Engineering Institute, CMU/SEI-89-TR-16.

[5] Graham, Marc H., "SQL Ada Module Description Language, Intermediate Version 3." 21 November 1991.

[6] LeClair, Allison and Susan Phillips, "A Prototype Implementation of the SQL Ada Module Extension (SAMe) Method." ACM Tri-Ada '90 Conference, Baltimore, MD, December 1990.

[7] *Reference Manual for the Ada Programming Language*. Ada Joint Program Office, 1983.

APPENDIX B

SAMeDL Development Environment Module Manager Top Level Design

SAMeDL Development Environment

Module Manager Top Level Design

Intermetrics, Inc.
6-January-1992
IR-VA-008-1

Published by
Intermetrics, Inc.
733 Concord Avenue, Cambridge, Massachusetts 02138

Table Of Contents

Chapter 1 Purpose

The purpose of this document is to describe the module manager for the SAMeDL Development Environment (SDE). A top level description of the SDE module manager will be provided, as well as descriptions of the user-SDE interface and the compiler-SDE interface.

The remainder of this document is organized as follows:

- Chapter 2, *Overview*, gives a brief overview of the SDE Module Manager.
- Chapter 3, *Data Structures*, outlines the physical disk data representation, internal representation and the data format.
- Chapter 4, *Operations*, documents the supported operations on the disk file and the internal representation.
- Chapter 5, *Module Manager Files*, lists the files present in the SDE Module Manager.
- Chapter 6, *User Interface*, documents the user interface commands for the user to interact with the SDE Module Manager.
- Appendix A, *Package Specifications*, presents the Ada package specifications for the interface routines and the definitions of the data structures.
- Appendix B, *Module Manager Commands*, includes manual pages for the user interface routines.

Chapter 2 Overview

The SDE module manager maintains the current dependency information for a SAMeDL design in a library; thus it is similar to an Ada library manager. The SDE module manager also acts as the manager of the repository (the library) for the SAMeDL source files containing the units and the files generated by the SAMeDL compilation.

The SDE module manager provides functionality to interact with both the user and the SAMeDL compiler. The compiler SDE-interface is in the form of procedure calls in the Ada programming language that the compiler can use, and the user SDE-interface is in the form of commands the user can type at the operating system prompt to execute various procedures. Operations the user might perform, for example, would be the creation and deletion of the SDE library, the generation of lists of units/files in the library, etc. The compiler would use SDE routines to add new information to the library as it compiles units, to extract dependency information about units, etc.

In a typical scenario, the SDE library is created by the user with the appropriate user interface command. Subsequent compilation of SAMeDL units modifies the library via the SAMeDL compiler SDE-interface. The user then uses other user SDE-interface commands to get information out of the library as well as modify the information present in it.

At the start of a SAMeDL compilation for a unit, the SDE data file is read into the compiler's internal data structures. Functions provided by the SDE module manager are used to perform this step. The use of internal data structures facilitates the quick retrieval of dependency information as well as the storing of new dependency information.

The internal representation of the library is tree-like, with each node in the tree corresponding to a file in the SDE library and containing information such as the file dependencies, creation time, related library files, etc. During compilation, new nodes may be added to this internal tree and new dependency arcs created to connect these nodes to previously existing nodes. The internal data structures are written to the SDE data file from the compiler at the end of each compilation using additional functions provided by the SDE module manager.

For example, consider the following SAMeDL code outlined below in Example 1.

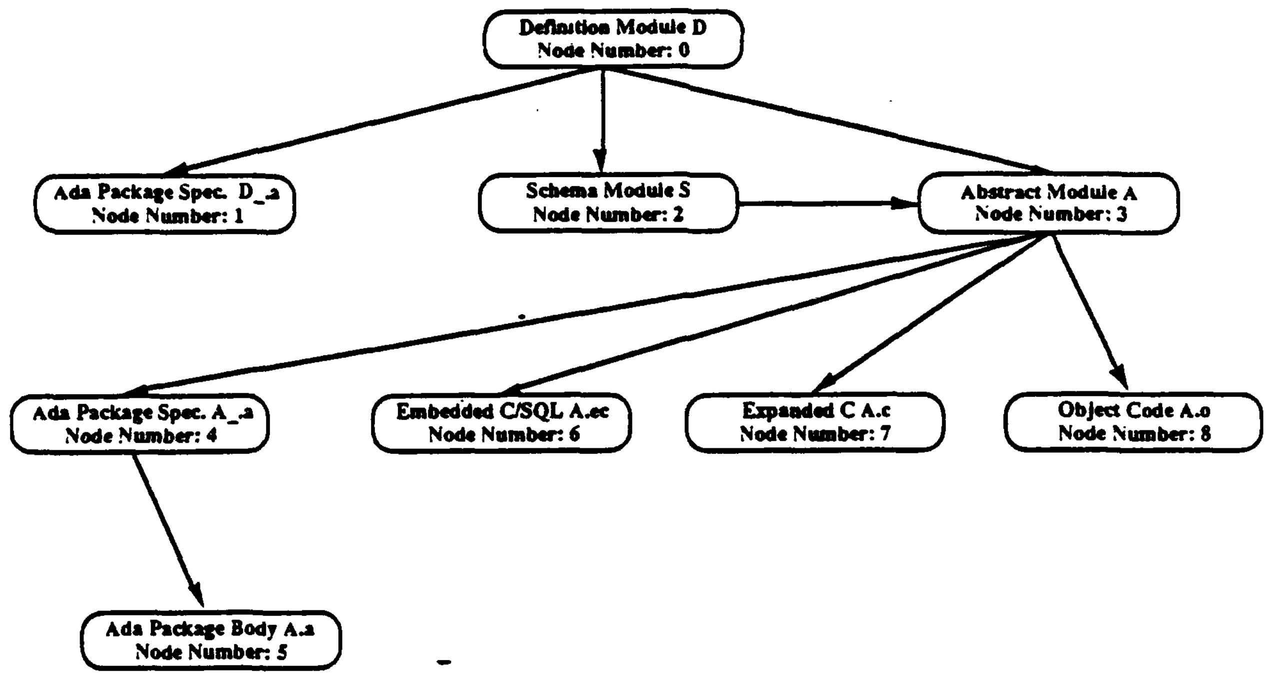
```
definition module D is
    .
    .
    .
end D;

with D; use D;
schema module S is
    .
    .
    .
end S;

with D; use D;
abstract module A is
        authorization S
    .
    .
    .
end A;
```

Example 1. SAMeDL Source Example.

The example contains a SAMeDL Definition Module D, a SAMeDL Schema Module S and a SAMeDL Abstract Module A. The compilation of the definition module D generates an Ada package specification named D_.a, the compilation of the schema module S generates no new files and the compilation of the abstract module A generates an Ada package specification named A_.a, an Ada package body named A.a, an embedded C/SQL source file A.ec, and then eventually, an expanded C file A.c and an object code file A.o. The library would like Figure 1 after the compilation is finished.



Unit Name	Node Number	Cares About (Node Numbers)	Cared About By (Node Numbers)
definition module *D*	0	*None*	1, 2, 3
ada package spec *D_.a*	1	0	*None*
schema module *S*	2	0	3
abstract module *A*	3	0, 2	4, 6, 7, 8
ada package spec *A_.a*	4	3	5
ada package body *A.a*	5	4	*None*
embedded C/SQL *A.ec*	6	3	*None*
expanded C *A.c*	7	3	*None*
object code *A.o*	8	3	*None*

Figure 1. State of SDE Library After Compilation of Example 1.

Chapter 3 Data Structures

The abstract data structure used by the SDE is tree-like, with each node on the tree corresponding to a SAMeDL unit or to a generated source file. Each node contains information about its dependencies on other nodes, the time it was created, the type of node, the related files, etc. This data structure is saved in a physical file in the library and also has an internal image that is used by the compiler and the user interface routines.

3.1 Physical File Structure

The physical file contains a series of records, each record containing the data for a single node in the internal representation of the dependency tree. The information in the disk file is in text format, that can be read/written using routines provided by the SDE module manager. File names for the files that the compiler generates are created using character prefixes and index numbers that are also saved in the disk data file.

3.2 Internal Representation

The internal representation of the dependency information is tree-like. Each node in the tree represents a file in the SAMeDL system, and has information about all nodes that are dependent on it and nodes that it depends on (called CaredAboutBy and CaredAbout arcs respectively). Each node also contains the time it was created, the external source file it was created from, the name of the source file saved in the library and the name of the library file that the generated code resides in.

Nodes are given a node numbers that uniquely identify them. This practice facilitates saving the tree to the designated disk file and reading it back because pointers do not need to be included in the disk file. It also facilitates the use of uniform data structures for the internal representation because variable length records do not need to be used. Instead, lists are maintained off each node that contain the node numbers of the nodes that the node depends upon, or is dependent upon.

3.3 Data Format

Both the records in the disk data file and the nodes in the internal representation have the same fields. The fields are:

Node Number	number of the node that specifies the unit
Node Type	the type of file this node points to
Unit Name	name of the compiled unit
Time Entered	time the unit was entered into the library
Library File	name of file saved in library
External File	pathname of file that the node was generated from
Cares About Arc Num	number of cares about arcs from this node
Cares About Arc List	list of cares about arcs from this node
Cared About By Arc Num	number of care about arcs to this node
Cared About By Arc List	list of care about arcs to this node

The records in the disk data file are written out in text form, one after the other with a special character separating each node. The disk data file also contains the current suffix numbers for the different types of files present in the library (described in Chapter 6).

In the internal representation, each node corresponds to a single record in the disk data file. New nodes may also be added during compilation by the SAMeDL compiler and their format is the same.

Chapter 4 Operations

This section describes the operations that are available to the compiler and the user interface programs for interacting with the two data representations (disk file, internal tree representation) that comprise the module manager.

All the procedures described below return a status variable signifying whether the procedure succeeded or failed. This parameter will not be explicitly mentioned below.

4.1 Operations on Disk File

The disk file that the data resides in is a pure text file, a format that can be changed easily if the current format is too cumbersome for the executable programs. When any process (compiler, user-interface program) communicates with the module manager, the library has to be locked. This locking prevents other instances of the SAMeDL executables that modify the library from modifying the data structures in use by the current SAMeDL process that has locked the library data file. The disk file is then read into the internal representation at the request of the current SAMeDL process that is modifying/reading the library, and then eventually written out after the process is done with its work. The library has to be unlocked before any other SAMeDL process may access the module manager library.

The syntax of the operations, including the names, parameters, errors generated, etc. may be found in the package specification for the package Disk_IO in Appendix A.

4.2 Operations on Internal Representation

The internal representation is a structure containing the nodes corresponding to the files in the SDE module manager library. The nodes are in the form of a tree, each node containing pointers to all nodes that it depends upon as well as pointers to nodes that depend upon it. Operations are provided to add nodes to this tree, create arcs connecting nodes, deleting nodes from the tree and walking the tree in breadth-first and depth-first fashion.

The operations for the manipulation of the internal representation are distributed over two packages. The first is the package Nodes_Package that contains the lower level operations that can be done on individual nodes. The other is the package Tree_Package that contains the operations that can be done on groups of nodes as they are connected. The syntax of the operations, including the names, parameters, errors generated, etc. may be found in the package specifications for these two packages in Appendix A.

Chapter 5 Module Manager Files

The file names in the following are Unix operating system dependent but can be changed easily for other operating systems.

The SAMeDL library contains the following files in it:

samedl.lib	directory of SDE module manager library
samedl.dat	file name of net data file
samedl.lock	lock file for SDE module manager library

It also contains the files that are generated by the SAMeDL compiler during the compilation of units. The filenames are in the following format where xxxxxxx corresponds to a number that is saved in the SDE module manager disk data file and is incremented each time a new file of a type is created. The number for each type of file is maintained independent of the others. The numbers may not be reused.

The first three files (Dxxxxxxx, Sxxxxxxx, Axxxxxxx) contain the modules that are extracted by the SAMeDL compiler from the user specified source file that is being compiled. They are pure text copies of the source, but only contain the module specified unlike the user specified file that may contain multiple SAMeDL modules in it.

The following files are saved in the module manager library:

Dxxxxxxx	source file for definition module
Sxxxxxxx	source file for schema module
Axxxxxxx	source file for abstract module
Pxxxxxxx	source file for Ada package specification
Bxxxxxxx	source file for Ada package body
Exxxxxxx	source file for embedded C file
Cxxxxxxx	source file for C file
Oxxxxxxx	object file

Chapter 6 User Interface

The user interface command names are Unix operating system dependent but can be changed easily. Further information about the command line options, arguments, defaults, and errors for the commands may be found in Appendix B.

sde.cleanlib	reinitialize library directory
sde.creatlib	creates a new SAMeDL library
sde.list	list units generated from a module
sde.ls	list compiled units
sde.rm	remove a SAMeDL source file or unit from a library
sde.rmlib	remove a SAMeDL library

Appendix A Package Specifications

The Ada package specifications for the SDE module manager follows:

```
--
-- globals_.a -
--
-- contains the global constant declarations required
-- throughout the module manager
--
package Global_Package is

-- Status Codes returned by functions/procedures
   type StatusType is (StatusOk, StatusError);

   Dir_Separator : constant String := "/";

   Library_Dir       : constant String := "samedl.lib";
   Database_File     : constant String := "samedl.dat";
   Lock_File         : constant String := "samedl.lock";
   Current_Library   : constant String := ".";

   RM_Command : constant String := "rm -rf";

end Global_Package;
```

```
--
-- disk_io_.a -
--
-- The disk file that the data resides in is a pure text file.  When any
-- process (compiler, user-interface process) communicates with the
-- module manager, the library has to be locked to other instances of
-- the processes.  This locking prevents the other processes from
-- modifying the data structures in use by the process already in the
-- library.  The locking and unlocking of the library is done using the
-- procedures in this package.  The disk file is read into the internal
-- structures using the procedures in this package.
--

with Global_Package; use Global_Package; -- Global types, constants
with Nodes_Package; use Nodes_Package; -- Types, constants

package Disk_IO is

  --
  -- LockLibrary(Status, LibraryPath) -- creates a lock file in the
  -- library if one does not already exist thereby prohibiting two
  -- people from modifying the library at the same time.  This should
  -- be done at the very start of processing a unit in the compiler.
  --
  -- Parameters:
  --   Status -- StatusOk if the procedure succeeds, StatusError otherwise
  --   LibraryPath -- pathname of the directory in which the module manager
  --     saves the data file.
  --
  procedure LockLibrary(
    Status      : in out StatusType;
    LibraryPath : in String);

  --
  -- UnlockLibrary(Status) -- delete the lock file opened by LockLibrary.
  -- Frees the library for use by other users.  This is the last thing
  -- that the compiler should do.  No more modifications to the library
  -- are allowed after an UnlockLibrary call without doing another
  -- LockLibrary call.
  --
  -- Parameters:
  --   Status -- StatusOk if the procedure succeeds, StatusError otherwise
  --
  procedure UnlockLibrary(
    Status      : in out StatusType);

  --
  -- ReadNodeFromKeyboard(Status, Node) -- reads a node from screen,
  --   For debugging purposes.
  --
  -- Parameters:
  --   Status -- StatusOk if the procedure succeeds, StatusError otherwise
  --   Node   -- pointer to the node read in from the keyboard.
  --
  procedure ReadNodeFromKeyboard(
    Status : in out StatusType;
    Node  : in out NodePtr);

  --
  -- WriteNodeToScreen(Status, Node) -- Writes a node to screen,
  --   For debugging purposes.
  --
  -- Parameters:
  --   Status -- StatusOk if the procedure succeeds, StatusError otherwise
  --   Node   -- pointer to the node to be printed on the screen.
  --
  procedure WriteNodeToScreen(
    Status : in out StatusType;
    Node   : in NodePtr);

  --
  -- ReadDiskData(Status, LibraryPath, Tree) -- reads the disk data file
  -- in the directory specified by LibraryPath into a tree and makes Tree
  -- point to it.
  --
  -- Parameters:
  --   Status -- StatusOk if the procedure succeeds, StatusError otherwise
  --   LibraryPath -- path for the directory in which the module manager
  --     saves the disk data file.
  --   Tree -- pointer to the root of the new tree created from reading
  --     the data in the disk data file.
  --
  procedure ReadDiskData(
    Status    : in out StatusType;
    LibraryPath : in String;
    Tree      : in out NodePtr);

  --
  -- WriteDiskData(Status, Tree, LibraryPath) -- writes the tree pointed
  -- to by Tree to the data disk file in the directory specified by
  -- LibraryPath after first making a copy if disk data file already exists
  -- in the module manager library directory specified by LibraryPath.
  --
  procedure WriteDiskData(
    Status    : in out StatusType;
    Tree      : in NodePtr;
    LibraryPath : in String);

end Disk_IO;
```

```
--
-- nodes_.a
--
-- The internal representation is a structure containing nodes corresponding
-- to the files in the module manager library directory. The node type is
-- declared in this package, the fields contain the information corresponding
-- to each file in the module manager library. The operations that manipulate
-- the fields in the nodes are declared in this package. The variables that
-- contain the current number suffixes for each kind of file in the library
-- are also maintained in this package.
--

with Global_Package; use Global_Package; -- Global types, constants.
with Calendar; -- for Time type.

package Nodes_Package is

-- Node Kinds available
  type NodeKind is (DefModule, SchemaModule, AbsModule, AdaPack,
    AdaPackBody, EmbeddedC, CSource, ObjectFile);

-- Pointer to strings used in the nodes.
  type StringPtr is access String;

-- CaresAbout node for the nodes that a node cares about (depends upon)
  type CaresAboutElement;
  type CaresAboutPtr is access CaresAboutElement;
  type CaresAboutElement is
    record
      Previous        : CaresAboutPtr; -- pointer to previous node in list
      Next            : CaresAboutPtr; -- pointer to next node in list
      NodeNumber      : Integer;     -- NodeNumber of node cared about by
                                     --   the node that has this in its
                                     --   cares about list.
    end record;

-- CaredAboutBy node for the nodes that a node is cared about by (dependent
-- upon).
  type CaredAboutByElement;
  type CaredAboutByPtr is access CaredAboutByElement;
  type CaredAboutByElement is
    record
      Previous        : CaredAboutByPtr; -- pointer to previous node in list
      Next            : CaredAboutByPtr; -- pointer to next node in list
      NodeNumber      : Integer;     -- NodeNumber of node that cares
                                     --   about the node that has this in
                                     --   its cared about by list.
    end record;

-- Node that is kept in a tree and in the external (physical) disk data file.
  type NodeElement;
type NodePtr is access NodeElement;
type NodeElement is
  record
    Previous          : NodePtr;        -- pointer to previous node in list
    Next              : NodePtr;        -- pointer to next node in list
    NodeNumber        : Integer;        -- Node number (unique) of node
    Kind              : NodeKind;       -- Kind of node
    Outdated          : Boolean;        -- True if node is outdated, else False
    UnitName          : StringPtr;      -- Unit Name of the unit that the node
                                        --   was compiled from.
    LibraryFile       : StringPtr;      -- File name of the file in the module
                                        --   manager library that contains the
                                        --   source text for the unit.
    ExternalFile      : StringPtr;      -- File name of the source text file
                                        --   that the unit for this node was
                                        --   compiled from.
    TimeEntered       : Calendar.Time; -- Time the node was created.
    NumCaresAbout     : Integer;        -- Number of nodes this node cares
                                        --   about.
    CaresAbout        : CaresAboutPtr;    -- List of nodes this node cares
                                          -- about.
    NumCaredAboutBy   : Integer;        -- Number of nodes this node is cared
                                        -- about by.
    CaredAboutBy      : CaredAboutByPtr; -- List of nodes this node is
                                         -- cared about by.
  end record;

-- Initialized when the database is read from the disk, incremented
-- each time a new node is created.
NextAvailNodeNumber : Integer;

-- The suffixes are initialized when the database is read from the
-- disk.  The file name for a kind is generated by a concatenation
-- of the prefix and the suffix and the suffix is incremented.
Definition_Module_Prefix     : constant String := "D";
Definition_Module_Suffix     : Integer;
Schema_Module_Prefix         : constant String := "S";
Schema_Module_Suffix         : Integer;
Abstract_Module_Prefix       : constant String := "A";
Abstract_Module_Suffix       : Integer;
Ada_Package_Prefix           : constant String := "P";
Ada_Package_Suffix           : Integer;
Ada_Package_Body_Prefix      : constant String := "B";
Ada_Package_Body_Suffix      : Integer;
Embedded_C_Prefix            : constant String := "E";
Embedded_C_Suffix            : Integer;
C_Source_Prefix              : constant String := "C";
C_Source_Suffix              : Integer;
Object_File_Prefix           : constant String := "O";
```

```
Object_File_Suffix              : Integer;

--
-- CreateNode(Status, Node, Kind, UnitName, ExternalFile) -- creates a
-- new node and initializes the NodeNumber, Kind, UnitName and ExternalFile
-- fields in the node.
--
-- Parameters:
--   Status - StatusOk if the procedure succeeds, StatusError otherwise.
--   Node   - pointer to the newly created node.
--   Kind   - the Kind of node to be created
--   UnitName - the name of the unit for which the node is to be created.
--   ExternalFile - the external file name of the file from which the
--     unit is being compiled.
--
procedure CreateNode(
  Status       : in out StatusType;
  Node         : out NodePtr;
  Kind         : in NodeKind;
  UnitName     : in String;
  ExternalFile : in String);


--
-- Empty (L) return Boolean -- returns True or False depending on whether
-- the list passed in is empty or not. This is an overloaded function,
-- and takes lists of three types: CaresAboutPtr, CaredAboutByPtr, and
-- NodePtr.
--
-- Parameters:
--   L -- pointer to the head of the list.
--
function Empty (L : in CaresAboutPtr) return Boolean;

function Empty (L : in CaredAboutByPtr) return Boolean;

function Empty (L : in NodePtr) return Boolean;


--
-- Append (Status, Node, List) -- appends Node to the List of nodes
-- passed in. This is an overloaded function, and takes Node and List
-- of the following types: CaresAboutPtr, CaredAboutByPtr, NodePtr.
--
-- Parameters:
--   Status -- StatusOk if the procedure succeeds, StatusError otherwise.
--   Node   -- pointer to the node to be appended.
--   List   -- pointer to the head of the list of nodes.
--
procedure Append (
  Status : in out StatusType;
  Node : in CaresAboutPtr;
  List : in out CaresAboutPtr);

procedure Append (
  Status : in out StatusType;
  Node : in CaredAboutByPtr;
  List : in out CaredAboutByPtr);

procedure Append (
    Status : in out StatusType;
    Node : in NodePtr;
    List : in out NodePtr);

  --
  -- Delete (Status, Node, List) -- deletes Node from the List passed in.
  -- This is an overloaded function and takes Node and List of the following
  -- types: CaresAboutPtr, CaredAboutByPtr, and NodePtr.
  --
  -- Parameters:
  --   Status -- StatusOk if the Delete succeeds, StatusError otherwise.
  --   Node   -- pointer to the node to be deleted from List
  --   List   -- pointer to the head of the list of nodes from which to
  --     delete the node.
  --
  procedure Delete (
    Status : in out StatusType;
    Node : in CaresAboutPtr;
    List : in out CaresAboutPtr);

  procedure Delete (
    Status : in out StatusType;
    Node : in CaredAboutByPtr;
    List : in out CaredAboutByPtr);

  procedure Delete (
    Status : in out StatusType;
    Node : in NodePtr;
    List : in out NodePtr);

  --
  -- CopyNode(Status, Node, NewNode) -- returns a copy of the node passed
  -- in. The next and the prev fields of the copied node are set to null.
  --
  -- Parameters:
  --   Status -- StatusOk if CopyNode succeeds, StatusError otherwise.
  --   Node   -- pointer to the node to be copied.
  --   NewNode -- pointer to a new node that is a copy of the node passed
  --     in.
  --
  procedure CopyNode(
    Status  : in out StatusType;
    Node    : in NodePtr;
    NewNode : out NodePtr);

end Nodes_Package;
```

```
--
-- tree_.a
--
-- contains the procedure specs and global variables to
-- manipulate trees of nodes.
--

with Global_Package; use Global_Package; -- Global types, constants.
with Nodes_Package; use Nodes_Package; -- Types, constants, required variables.

package Tree_Package is

  -- The global tree that the data is read into. Used by user-interface
  -- commands. Unnecessary, can be removed if the other packages declare
  -- their own tree variable.
  GlobalLibTree      : NodePtr;

  --
  -- AddNodeToTree(Status, Node, Tree) -- adds the node to the library tree
  -- Tree. Not different from Nodes_Package.Append at the present time
  -- because the Tree is not tree structured.
  --
  -- Parameters:
  --   Status -- StatusOk if the procedure succeeds, StatusError otherwise.
  --   Node   -- pointer to the node to be appended to the tree.
  --   Tree   -- pointer to the root of the tree to which the node is to
  --             be added.
  --
  procedure AddNodeToTree(
    Status : in out StatusType;
    Node   : in NodePtr;
    Tree   : in out NodePtr);

  --
  -- DeleteNodeFromTree(Status, Node, Tree) -- deletes the node from the
  -- tree passed in. Not different from Nodes_Package.Delete at the present
  -- time because the Tree is not tree structured.
  --
  -- Parameters:
  --   Status -- StatusOk if the procedure succeeds, StatusError otherwise.
  --   Node   -- pointer to the node to be deleted from the tree.
  --   Tree   -- pointer to the root of the tree from which the node is to
  --             be deleted.
  --
  procedure DeleteNodeFromTree(
    Status : in out StatusType;
    Node   : in NodePtr;
    Tree   : in out NodePtr);

  --
  -- AddCaresAboutArc(Status, From, To) -- add a cares about arc
  -- from the From node to the To node.
```

```
--
-- Parameters:
--  Status -- StatusOk if the procedure succeeds, StatusError otherwise.
--  From   -- pointer to the node from which the arc emanates.
--  To     -- pointer to the node to which the arc points.
--
procedure AddCaresAboutArc(
  Status  : in out StatusType;
  From    : in out NodePtr;
  To      : in out NodePtr);

--
-- AddCaresAboutArc(Status, Node, Kind, UnitName) -- add a cares about arc
-- from node to the node for unitname, kind.
--
-- Parameters:
--  Status -- StatusOk if the procedure succeeds, StatusError otherwise.
--  From   -- pointer to the node from which the arc emanates.
--  Kind   -- Kind of node to which the arc is to point.
--  UnitName -- the name of the Unit to which the arc is to point.
--  Tree   -- pointer to the tree in which the nodes are present.
--
procedure AddCaresAboutArc(   -
  Status  : in out StatusType;
  From    : in out NodePtr;
  Kind    : in NodeKind;
  UnitName : in String;
  Tree    : in out NodePtr);

--
-- AddCaredAboutByArc(Status, From, To) -- add a cared about by arc
-- from the From node to the To node.
--
-- Parameters:
--  Status -- StatusOk if the procedure succeeds, StatusError otherwise.
--  From   -- pointer to the node from which the arc emanates.
--  To     -- pointer to the node to which the arc points.
--
procedure AddCaredAboutByArc(
  Status  : in out StatusType;
  From    : in out NodePtr;
  To      : in NodePtr);

--
-- AddCaredAboutByArc(Status, Node, Kind, UnitName, Tree) -- add a cared
-- about by arc from node to the node for unitname, kind, in Tree
--
-- Parameters:
```

```
-- Status -- StatusOk if the procedure succeeds, StatusError otherwise.
-- From   -- pointer to the node from which the arc emanates.
-- Kind   -- Kind of node to which the arc is to point.
-- UnitName -- the name of the Unit to which the arc is to point.
-- Tree   -- pointer to the tree in which the nodes are present.
--
procedure AddCaredAboutByArc(
  Status   : in out StatusType;
  From     : in out NodePtr;
  Kind     : in NodeKind;
  UnitName : in String;
  Tree     : in out NodePtr);


--
-- CopyTree(Status, Tree, NewTree) -- returns a pointer to the head of a
-- tree that is a copy of the tree passed in.
--
-- Parameters:
-- Status -- StatusOk if the procedure succeeds, StatusError otherwise
-- Tree -- pointer to the root of the tree to be copied
-- NewTree -- pointer to the root of the copied tree.
--
procedure CopyTree(
  Status  : in out StatusType;
  Tree    : in NodePtr;
  NewTree : out NodePtr);


--
-- FindNode(Status, NodeNumber, Tree, Node) -- find the nodes with
-- nodenumber equal nodenumber and returns a pointer to the node.
--
-- Parameters:
-- Status -- StatusOk if the procedure succeeds, StatusError otherwise
-- NodeNumber -- Nodenumber of the node to find
-- Tree -- pointer to the root of the tree to search
-- Node -- pointer to the found node.
--
procedure FindNode(
  Status     : in out StatusType;
  NodeNumber : in Integer;
  Tree       : in NodePtr;
  Node       : out NodePtr);


--
-- FindNode(Status, Kind, UnitName, Node) -- finds the nodes with
-- unit_name and kind and returns a pointer to the node.
--
-- Parameters:
-- Status -- StatusOk if procedured succeeds, StatusError otherwise
-- Kind   -- Kind of the node to find
-- UnitName -- name of the unit that was compiled for the node to find.
-- Tree   -- pointer to the root of the tree to search for the node.
-- Node   -- pointer to the found node.
--
procedure FindNode(
  Status  : in out StatusType;
  Kind    : in NodeKind;
  UnitName : in String;
  Tree     : in out NodePtr;
  Node     : out NodePtr);

--
-- FindUnitOutdatedness - finds if a Kind/UnitName is out of date. If
-- it is then the Outdated parameter is set to true, and a list of all
-- outdated cared about nodes is returned in List.
--
-- Parameters:
--   Status -- StatusOk if procedure succeeds, StatusError otherwise
--   LibraryTree -- Tree in which to search for nodes and follow arcs.
--   Kind -- Kind of the node to find outdatedness of.
--   UnitName -- Unit name of the Unit to check outdatedness of.
--   List -- list of nodes that are outdated and thus cause the unit being
--     checked to be outdated.
--
procedure FindUnitOutdatedness(
  Status      : in out StatusType;
  LibraryTree : in out NodePtr;
  Kind        : in NodeKind;
  UnitName    : in String;
  Outdated    : out Boolean;
  List        : in out NodePtr);

--
-- BreadthFirstWalk(Status, Tree, CaresAboutList, Head) -- walks
-- the cares or cared about arcs given and returns a list
-- of nodes. Overloaded function, walks the CaresAboutList, or the
-- CaredAboutByList.
--
-- Parameters:
--   Status -- StatusOk if the procedure succeeds, StatusError otherwise
--   Tree -- pointer to the root of the tree to walk
--   CaresAboutList/CaredAboutByList -- pointer to the head of the list
--     or nodes that care to be walked.
--   Head -- pointer to the head of a list of nodes that were visited
--     during the walk.
--
procedure BreadthFirstWalk(
  Status         : in out StatusType;
  Tree           : in NodePtr;
  CaresAboutList : in CaresAboutPtr;
  Head           : out NodePtr);

procedure BreadthFirstWalk(
  Status           : in out StatusType;
  Tree             : in NodePtr;
  CaredAboutByList : in CaredAboutByPtr;
  Head             : out NodePtr);

--
```

```
-- DepthFirstWalk(Status, Tree, CaresAboutList, Head) -- walks the cares
-- about/cared about by arcs and returns a list of nodes walked.
-- Overloaded function, walked CaresAbout arcs, and CaredAboutBy arcs.
--
-- Parameters:
--   Status -- StatusOk if the procedure succeeds, StatusError otherwise
--   Tree   -- pointer to the tree to walk the nodes on.
--   CaresAboutList/CaredAboutByList --  list of nodes to walk
--   Head -- list of nodes visited during the walk.
--
procedure DepthFirstWalk(
  Status         : in out StatusType;
  Tree           : in NodePtr;
  CaresAboutList : in CaresAboutPtr;
  Head           : in out NodePtr);

procedure DepthFirstWalk(
  Status           : in out StatusType;
  Tree             : in NodePtr;
  CaredAboutByList : in CaredAboutByPtr;
  Head             : in out NodePtr);

end Tree_Package;
```

Appendix B Module Manager Commands

The man pages for the SDE module manager commands follow:

sde.cleanlib - reinitialize library directory

Syntax

```
sde.cleanlib [pathname]
```

Description

sde.cleanlib will empty the directory samedl.lib present in the directory specified by pathname of all files, and reinitialize the disk data file. The default pathname is the current directory.

Examples

The following sequence of commands cleans and reinitializes the SDE module manager library in the directory /home/samedl.

```
$ cd /home/samedl

$ sde.cleanlib
```

The following command does the same thing:

```
$ sde.cleanlib /home/samedl
```

Diagnostics

The user is prompted to confirm the cleaning of the library. An error message is generated if the samedl.lib directory does not exist in the pathname specified (or current directory if the pathname option is not specified).

sde.creatlib - make a library directory

Syntax

```
sde.creatlib [pathname]
```

Description

sde.creatlib creates and initializes a new SAMeDL library directory. It creates a directory named samedl.lib for the library in the directory specified by the pathname option. If the pathname option is not used, the current directory is the default. **sde.creatlib** creates a disk data file named **samedl.dat** in the new directory. It initializes the disk data file to be empty and sets the information fields to initial states.

Examples

The following sequence of commands creates a new SDE module manager library in the directory /home/samedl.

```
$ cd /home/samedl

$ sde.creatlib
```

The following command does the same thing:

```
$ sde.creatlib /home/samedl
```

Diagnostics

The user is prompted to confirm the creation of the SAMeDL module manager library in the directory specified by the pathname option (the current directory is the pathname option is not specified). An error message is generated if the creation of the library is unsuccessful.

sde.list - list units/source files generated from the SAMeDL unit specified.

Syntax

sde.list [options] [unit_name]

Options

-a	(Ada) List only units with Ada package or Ada package body types.
-f source_file	(file) consider the SAMeDL units declared in source_file that the user created/compiled into the SDE module manager library as parent units to find order information.
-L pathname	(library) Operate in SDE module manager library present in the directory specified in pathname (the current directory is the default).
-l	(list) List the unit name, unit kind unit date, library source file, and external file.

Description

sde.list provides a list of units in the library that were generated from the compilation of the SAMeDL unit specified in the command line. All the units in the library are listed if no unit is specified. This command would be useful for creating script files for automatic compilation of files into the user's ada library.

Examples

The following sequence of commands lists the units generated by the compilation of the SAMeDL module named abstract_mod in the SDE module manager library present in the directory /home/samedl.

```
$ cd /home/samedl

$ sde.list abstract_mod
```

The following command does the same thing:

```
$ sde.list -L /home/samedl abstart_mod
```

The following sequence of commands lists only the Ada package specs and bodies that were generated from the units in the file user.sme that the user compiled into the library. The SDE module manager library is assumed to be in the directory /home/samedl.

```
$ cd /home/samedl

$ sde.list -a -f user.sme
```

The following command does the same thing:

```
$ sde.list -L /home/samedl -a -f user.sme
```

Diagnostics

An error message is generated if the SDE module manager library does not exist in the directory specified by the -L option (or in the current directory is the -L option is unspecified). Another error message is generated if the SDE module manager library is locked by another process.

sde.ls - list compiled units

Syntax

sde.ls [options] [unit_name]

Options

-a	(all) List all units visible in the library.
-f source_file	(file) List only units found in the user created/compiled source file.
-L pathname	(library) Operate in SDE module manager library present in the directory specified by pathname (the current directory is the default).
-l	(long) List unit, unit_type, library entry date, source file name, library file name.

Description

sde.ls provides a list of the SAMeDL units compiled in the SDE module manager library in the current or specified user directory. Options are provided to give more or less extensive information, or to provide a list of compiled units occurring in specified source files. Providing the unit name of a unit gives information only about the specified unit.

Examples

The following sequence of commands lists the units in the SDE module manager library present in the directory /home/samedl.

```
$ cd /home/samedl

$ sde.ls
```

The following command does the same thing:

```
$ sde.ls -L /home/samedl
```

The following sequence of commands lists complete information about the unit abstract_mod present in the SDE module manager library present in the directory /home/samedl.

```
$ cd /home/samedl
$ sde.ls -l
```

The following command does the same thing:

```
$ sde.ls -L /home/samedl -l
```

Diagnostics

An error message is generated if the SDE module manager library does not exist in the directory specified by the -L option (or in the current directory is the -L option is unspecified). Another error message is generated if the SDE module manager library is locked by another process.

sde.rm - remove unit and library information

Syntax

sde.rm [options] unit_name

sde.rm [options] source_file

Options

-L pathname	(library) Operate in SDE module manager library in the directory specified by pathname (the current directory is the default).
-V	(verify) verify the removal of each unit.
-v	(verbose) list the units as they are removed.

Description

sde.rm removes all information associated with the named unit or file. When unit_name is specified, the corresponding files in the library are removed.

When source_file is specified, the units in the user created/compiled file as well as the corresponding files in the library are removed.

Examples

The following sequence of commands removes the unit abstract_mod from the SDE module manager library present in the directory /home/samedl.

```
$ cd /home/samedl

$ sde.rm abstract_mod
```

The following command does the same thing:

```
$ sde.rm -L /home/samedl abstract_mod
```

Diagnostics

An error message is generated if the SDE module manager library does not exist in the directory specified by the -L option (or in the current directory is the -L option is unspecified). Another error message is generated if the SDE module manager library is locked by another process.

If the verify option is specified, the user is prompted to verify the removal of the specified units.

sde.rmlib - remove SAMeDL library

Syntax

sde.rmlib [options]

Options

-L pathname	(library) Operate in SDE module manager library in the directory specified by pathname (the current directory is the default).
-v	(verbose) list the units as they are removed.

Description

sde.rmlib removes all information in the SDE module manager library in the directory specified by the -L option (the current directory is the default). It deletes all the files in the SDE module manager library directory, and the removes the directory.

Examples

The following sequence of commands removes the SDE module manager library present in the directory /home/samedl.

```
$ cd /home/samedl

$ sde.rmlib
```

The following command does the same thing:

```
$ sde.rmlib -L /home/samedl
```

Diagnostics

The user is prompted to confirm the removal of the library. An error message is generated if the SDE module manager library does not exist in the directory specified by the -L option (or in the current directory is the -L option is unspecified). Another error message is generated if the SDE module manager library is locked by another process.

APPENDIX G

CSU Test Cases

Promotion Standing List Removal Action (Test 01) This test procedure removes a soldier from the E5-E6 Promotion Standing List whose grade is E4 (rank is SPC). A Removal From Local Recommended List Memorandum, PCN AAA-034, is generated.

ACTION	RESPONSE	VERIFICATION
1. Enter <P> at SIDMENU.	SID0010 appears on the screen.	
2. Enter <S> at SID0010.	M06020010 appears on the screen.	
3. Enter <L> at M06020010.	M06020310 appears on the screen.	
4. Enter <R> at M06020310.	SYST0001 appears on the screen.	
5. Enter "979-66-2499" in SSN.	Data is accepted.	
6. Enter "SUMMER" in Name and press <F2>.	S06020320 appears on the screen. SSN, Name, UIC, Promotion Rank Considered, and Promotion Indicator are displayed with values filled in.	
7. Enter "P" in Removal Reason Indicator.	Data is accepted.	
8. Enter "AAA" in Authentication Indicator and press <F2>	SYST0002 appears on the screen.	
9. Enter <V> at SYST002.	The report, PCN AAA-034, appears on the screen.	
10. Press <F4>.	SYST0002 appears on the screen.	
11. Enter <P> at SYST0002.	The report, PCN AAA-034, is printed.	

<u>Promotion Standing List Removal Action (Test 02)</u> This test procedure removes a soldier from the E5-E6 Promotion Standing List whose grade is E5 (rank is SGT). A Removal From Local Recommended List Memorandum, PCN AAA-034, is generated.

ACTION	RESPONSE	VERIFICATION
1. Enter <P> at SIDMENU.	SID0010 appears on the screen.	
2. Enter <S> at SID0010.	M06020010 appears on the screen.	
3. Enter <L> at M06020010.	M06020310 appears on the screen.	
4. Enter <R> at M06020310.	SYST0001 appears on the screen.	
5. Enter "979-88-0999" in SSN.	Data is accepted.	
6. Enter "BOLLIN" in Name and press <F2>.	S06020320 appears on the screen. SSN, Name, UIC, Promotion Rank Considered, and Promotion Indicator are displayed with values filled in.	
7. Enter "H" in Removal Reason Indicator.	Data is accepted.	
8. Enter "ABX" in Authentication Indicator and press <F2>.	SYST0002 appears on the screen.	
9. Enter <V> at SYST002.	The report, PCN AAA-034, appears on the screen.	
10. Press <F4>.	SYST0002 appears on the screen.	
11. Enter <P> at SYST0002.	The report, PCN AAA-034, is printed.	

Promotion Standing List Removal Action (Test_03) This test procedure verifies that when removing a soldier from the E5-E6 Promotion Standing List whose grade is E5 (rank is SGT): pop-up windows are available for fields with pop-up help capability; appropriate user and error messages are displayed; and help screens are available for each screen and menu. A Removal From Local Recommended List Memorandum, PCN AAA-034, is generated.

ACTION	RESPONSE	VERIFICATION
1. Enter <P> at SIDMENU.	SID0010 appears on the screen.	
2. Enter <S> at SID0010.	M06020010 appears on the screen.	
3. Enter <L> at M06020010.	M06020310 appears on the screen.	
4. Press <SHIFT-F1>.	Help screen M0602H310 appears on the screen. There is a description of each menu choice: Removal Action and Reinstatement Action.	
5. Press <F4>.	M06020310 appears on the screen.	
6. Enter <B> at M06020310.	User message 00011, INVALID MENU CHOICE. TRY AGAIN, is displayed.	
7. Enter <R> at M06020310.	SYST0001 appears on the screen.	
8. Enter "979-49-7499" in SSN.	Data is accepted.	
9. Enter "RUMPH" in Name and press <F2>.	S06020320 appears on the screen. SSN, Name, UIC, Promotion Rank Considered, and Promotion Indicator are displayed with values filled in.	
10. Enter "U" in Removal Reason Indicator.	Data is accepted.	

ACTION	RESPONSE	VERIFICATION
11. Enter "ABX" in Authentication Indicator and press <F2>.	User message 01032, INVALID DATA, is displayed. Cursor is in the Removal Reason Indicator field.	
12. Press <CODE-F1>.	Pop-up window for Removal Reason Indicator is displayed.	
13. Highlight value "S" and press <CR>.	Removal Reason Indicator field is filled in with "S" for a value. Cursor is in the Removal Reason Indicator field.	
14. Press <CR>.	Cursor moves to the Authentication Indicator Code field.	
15. Enter "XXX" in Authentication Indicator and press <F2>.	User message 00411, INVALID AUTHENTICATION INDICATOR, is displayed. Cursor is in the Authentication Indicator field.	
16. Press <CODE-F1>.	Pop-up window for Authentication Indicator is displayed.	
17. Highlight Authentication Indicator value "ABA" and press <CR>.	Authentication Indicator field is filled in with "ABA" for a value. Cursor is in the Authentication Indicator field.	
18. Press <F2>.	SYST0002 appears on the screen.	
19. Enter <V> at SYST0002.	The report, PCN AAA-034, appears on the screen.	
20. Press <F4>.	SYST0002 appears on the screen.	
21. Enter <P> at SYST0002.	The report, PCN AAA-034, is printed.	

ACTION	RESPONSE	VERIFICATION
22. Enter <X> at SYST002.	M06020310 appears on the screen.	

Promotion Recommendation Actions (Test 01). This test procedure performs an initial calculation of administrative points for an individual soldier whose current grade is E4 (rank is SPC). A DA Form 3855-E, PCN AAA-209, is generated.

ACTION	RESPONSE	VERIFICATION
1. Enter <P> at SIDMENU.	SID0010 appears on the screen.	
2. Enter <S> at SID0010.	M06020010 appears on the screen.	
3. Enter <P> at M06020010.	M06020080 appears on the screen.	
4. Enter <I> at M06020080.	SYST0001 appears on the screen.	
5. Enter "979-66-9699" in SSN.	Data is accepted.	
6. Enter "MCKISS" in name and press <F2>.	S06020100 appears on the screen. SSN, Name, UIC, and Rank are displayed with values filled in.	
7. Enter "175" in Duty Performance Points.	Data is accepted.	
8. Enter "125" in Military Education Points.	Data is accepted.	
9. Enter "85" in Civilian Education Points and press <CR>.	Data is accepted.	
10. Enter "AAA" in Authentication Indicator and press <F2>.	SYST0002 appears on the screen.	
11. Enter <V> at SYST0002.	The report, PCN AAA-209, appears on the screen.	
12. Enter <P> at SYST0002.	The report, PCN AAA-209, is printed.	

Promotion Recommendation Actions (TEST 02). This test procedure performs an initial calculation of administrative points for an individual soldier whose current grade is E5 (rank is SGT). A DA Form 3855-E, PCN AAA-209, is generated.

ACTION	RESPONSE	VERIFICATION
1. Enter <P> at SIDMENU.	SID0010 appears on the screen.	
2. Enter <S> at SID0010.	M06020010 appears on the screen.	
3. Enter <P> at M06020010.	M06020080 appears on the screen.	
4. Enter <I> at M06020080.	SYST0001 appears on the screen.	
5. Enter "979-12-9399" in SSN.	Data is accepted.	
6. Enter "SCOTT" in Name and press <F2>.	S06020100 appears on the screen. SSN, Name, UIC, and Rank are displayed with values filled in.	
7. Enter "150" in Duty Performance Points.	Data is accepted.	
8. Enter "100" in Military Education Points.	Data is accepted.	
9. Enter "75" in Civilian Education Points and press <CR>.	Data is accepted.	
10. Enter "AAA" in Authentication Indicator and press <F2>.	SYST0002 appears on the screen.	
11. Enter <V> at SYST0002.	The report, PCN AAA-209, appears on the screen.	
12. Press <F4>.	SYST0002 appears on the screen.	
13. Enter <P> at SYST0002.	The report, PCN AAA-209, is printed.	

Promotion Recommendation Actions (TEST 03). This test procedure verifies that when performing an initial calculation of administrative points for an individual soldier whose current grade is E4 (rank is SPC); pop-up windows are available for fields with pop-up help capability; appropriate user and error messages are displayed; and help screens are available for each screen and menu. A DA Form 3855-E, PCN AAA-209, is generated.

ACTION	RESPONSE	VERIFICATION
1. Enter <P> at SIDMENU.	SID0010 appears on the screen.	
2. Enter <S> at SID0010.	M06020010 appears on the screen.	
3. Enter <P> at M06020010.	M06020080 appears on the screen.	
4. Enter <I> at M06020080.	SYST0001 appears on the screen.	
5. Enter "979-02-5599" in SSN.	Data is accepted.	
6. Enter "JACKSO" in Name and press <F2>.	S06020100 appears on the screen.	
7. Press <F1>.	Help screen S0602H100 appears on the screen. There is a description of the entry fields: Duty Performance Points, Military Education Points, Civilian Education Points, and Authentication Indicator.	
8. Press <F4>.	M06020100 appears on the screen.	
9. Enter "ABOVE" in Duty Performance Points.	Key strokes are ignored.	
10. Enter "201" in Duty Performance Points.	Data is accepted.	
11. Enter "150" in Military Performance Points.	Data is accepted.	

ACTION	RESPONSE	VERIFICATION
12. Enter "100" in Civilian Education Points.	Data is accepted.	
13. Enter "AAA" in *Authentication Indicator* field and press <F2>.	User message 00803, MAXIMUM POINTS ALLOWED IS 200 FOR ACTIVE DUTY PERFORMANCE, is displayed. Cursor is in the Duty Performance Points field.	
14. Enter "200" in Duty Performance Points.	Data is accepted.	
15. Enter "Below" in Military Performance Points.	Key strokes are ignored.	
16. Enter "151" in Military Performance Points and press <F2>.	User message 00799, MAXIMUM POINTS ALLOWED IS 150 FOR ACTIVE MILITARY EDUCATION, is displayed. *Cursor* is in the Military Education Points field.	
17. Enter "150" in Military Performance Points.	Data is accepted.	
18. Enter "Average" in Civilian Education Points.	Key strokes are ignored.	
19. Enter "101" in Civilian Education Points and press <F2>.	User message 00801, MAXIMUM POINTS ALLOWED IS 100 FOR ACTIVE CIVILIAN EDUCATION, is displayed. Cursor is in the Civilian Education Points field.	
20. Enter "100" in Civilian Education Points.	Data is accepted.	

ACTION	RESPONSE	VERIFICATION
21. Enter "NBA" in Authentication Indicator field and press <F2>.	User message 00411, INVALID AUTHENTICATION INDICATOR, is displayed. Cursor is in the Authentication Indicator field.	
22. Press <CODE-F1>.	Pop-up window for Authentication Indicator is displayed.	
23. Highlight Authentication Indicator "AAA" and press <CR>.	Authentication Indicator field is filled in with "AAA" for a value. Cursor is in the Authentication Indicator field.	
24. Press <F2>.	SYST0002 appears on the screen.	
25. Enter <V> at SYST0002.	The report, PCN AAA-209, appears on the screen.	
26. Press <F4>.	SYST0002 appears on the screen.	
27. Enter <P> at SYST0002.	The report, PCN AAA-209, is printed.	